НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр Коваль
«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Геометричне моделювання в інформаційних системах»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «Мікросервіс розрахунку мінімального габаритного циліндру 3D моделі»

Виконала:
студентка IV курсу, групи ТР-61
Свєтла Лоліта Вікторівна _____

Керівник:
доцент, к.т.н,
Демчишин А.А. _____

Рецензент:
професор, д.т.н., професор,
Несвідомін В.М. _____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.
Студентка _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Геометричне моделювання в інформаційних системах

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
“___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

Світлій Лоліті Вікторівні
(прізвище, ім’я, по батькові)

1. Тема роботи “Мікросервіс розрахунку мінімального габаритного циліндру 3D моделі”

керівник роботи к.т.н., доцент Демчишин Анатолій Анатолійович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від “25” травня 2020р. № **1268-с**

2. Строк подання студентом роботи “15” червня 2020р.

3. Вихідні дані до роботи завантажена модель, розрахунок мінімального радіусу і висоти габаритного циліндру моделі, вікно з результатами розрахунку, завантаженим циліндром та хмарою точок.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) провести огляд сучасного стану індустрії систем, які включають в себе алгоритми розрахунку мінімальних обмежуючих фігур і тіл для хмари точок, вибрати інструменти розробки, розробити алгоритм розрахунку мінімального габаритного циліндру, провести реалізацію програмної системи, що працює на базі розробленого алгоритму

5. Перелік ілюстративного матеріалу 1. Актуальність роботи. 2. Мета і задачі роботи. 3. Існуючі програмні системи. 4. Інструменти реалізації. 5. Хмарний сервіс. 6. Мікросервісна архітектура веб-додатків. 7. Схема взаємодії мікросервісів. 8. Архітектура розробленого веб-додатку. 9. Алгоритм Вельцля. 10. Складність реалізованого алгоритму. 11. Приклад роботи програми. 12. Висновки.

6. Дата видачі завдання “_15_”____жовтня____2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	15.10.19	
2.	Вивчення та аналіз задачі	16.10.19 – 31.12.19	
3.	Розробка архітектури та загальної структури системи	01.01.20 – 25.03.20	
4.	Розробка структур окремих підсистем	26.03.20 – 12.04.20	
5.	Програмна реалізація системи	13.04.20 – 19.05.20	
6.	Оформлення пояснювальної записки	20.05.20 – 10.06.20	
7.	Захист програмного продукту	05.06.20	
8.	Передзахист	05.06.20	
9.	Захист	15.06.20	

Студентка _____ _____Світла Л.В._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Демчишин А.А._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою роботи є розробка веб-системи, що вирішує задачу пошуку мінімального габаритного циліндру 3D моделі. Для вирішення даної задачі було прийнято рішення використати мікросервісну архітектуру і хмарні технології. Основною мовою програмування розробленої системи є JavaScript. Для реалізації серверної частини веб-системи обрана платформа Node.js. Клієнтська частина системи реалізована за допомогою HTML, CSS і JavaScript. Обмін інформацією між мікросервісами і клієнтською частиною програми відбувається за допомогою REST API. Пошук мінімального габаритного циліндру побудований на основі алгоритму Вельцля.

Записка містить 73 сторінки, 26 рисунків, 10 посилань і 1 таблицю.

Ключові слова: JavaScript, Node.js, хмарний сервіс, Three.js, STL, складність алгоритму.

ABSTRACT

The purpose of the work is to develop a web system that solves the problem of the minimum overall cylinder of the 3D model. To solve this problem, it was decided to use microservice architecture and cloud technologies. The main programming language of the developed system is JavaScript. The Node.js platform was chosen to implement the server part of the web system. The client part of the system is implemented using HTML, CSS and JavaScript. The exchange of information between the microservices and the client part of the program takes place using the REST API. The search of the minimum overall cylinder is based on the Welzl algorithm.

The note contains 73 pages, 26 images, 10 references and 1 table.

Keywords: JavaScript, Node.js, cloud service, Three.js, STL, algorithm complexity.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

3D – 3-dimensional, тривимірний.

API – Application Programming Interface – програмний інтерфейс додатків.

CSS – Cascading Style Sheets, каскадні таблиці стилів – формальна мова опису зовнішнього вигляду документа.

HTML – HyperText Markup Language, мова гіпертекстової розмітки – стандартизована мова розмітки документів у Всесвітній павутині.

Node.js – програмне середовище, що дозволяє використовувати мову JavaScript як мову загального призначення.

Three.js – бібліотека для роботи з тривимірною графікою.

REST API – архітектурний стиль взаємодії компонентів розподіленого додатку в мережі.

STL – (від англ. stereolithography) – формат файлу, що широко використовується для зберігання тривимірних моделей об’єктів.

ВСТУП

На сьогоднішній день світ та технології не стоять на місці. Зі збільшенням населення виникає потреба в збільшенні об’ємів виробництва. Завдяки інформаційним технологіям, що швидко розвиваються, підприємства мають можливість збільшувати виробничі потужності.

В умовах високого темпу виробництва на підприємствах виникає питання оптимізації використання ресурсів, зокрема матеріалу для виготовлення деталей виробів. Особливо гостро постає це питання перед промисловцями, що виготовляють нестандартні деталі на замовлення, адже їм складніше мінімізувати використання матеріалу, а інколи навіть не завжди є можливість визначити можливість виготовлення замовленої деталі, оскільки заготівка занадто мала.

Метою даної роботи є створення веб-системи на базі мікросервісної архітектури і середовища Node.js для вирішення задачі мінімального габаритного циліндру 3D моделі. Актуальність розробленої системи визначається необхідністю розробки алгоритму розрахунку мінімального циліндру, що обмежує хмару точок. Для досягнення мети роботи необхідно виконати наступні задачі:

- провести огляд сучасного стану індустрії систем, які включають в себе алгоритми розрахунку мінімальних обмежуючих фігур і тіл для хмари точок;
- вибрати інструменти розробки;
- розробити алгоритм розрахунку мінімального габаритного циліндру;
- провести реалізацію програмної системи, що працює на базі розробленого алгоритму.

Об’єктом роботи є розробка програмної системи на основі алгоритму розрахунку мінімального габаритного циліндру 3D моделі.

Предметом роботи є теоретичні засади алгоритмів розрахунку мінімального габаритного циліндру.

Роботу виконано в контексті НДР 0119U103633 Аналіз і візуалізація геометричних та геоінформаційних даних.

На даний момент актуальним напрямком є реалізація алгоритму для обчислення мінімального габаритного циліндру. Для вирішення поставленої задачі можна застосувати алгоритм лінійної складності для розрахунку мінімальної обмежуючої окружності, розроблений Еммеріхом Вельцлем. Саме він буде використовуватися в роботі та послугує основою для розрахунку мінімального циліндру, що обмежує хмару точок.

Звіт має наступну структуру:

У першому розділі описано постановку задачі і огляд існуючих подібних систем.

У другому розділі описано засоби реалізації системи.

У третьому розділі наведено опис програмної реалізації.

У четвертому розділі описано сценарій роботи користувача з програмою.

1. ПОСТАНОВКА ЗАДАЧІ І АНАЛІЗ СТАНУ ІСНУЮЧИХ СИСТЕМ, ЯКІ ВКЛЮЧАЮТЬ В СЕБЕ АЛГОРИТМИ РОЗРАХУНКУ МІНІМАЛЬНИХ ОБМЕЖУЮЧИХ ФІГУР І ТІЛ ДЛЯ ХМАРИ ТОЧОК

Метою роботи є реалізація веб-системи для побудови мінімального габаритного циліндру 3D моделі, що складається з хмари точок. Дане програмне забезпечення буде вирішувати задачу оптимізації використання матеріалу на виробництві, а також задачу оцінки можливості виготовлення виробу певного розміру. Саме тому потенційними користувачами даної системи будуть підприємці, які працюють з заготівками циліндричної форми, зокрема виробники ортодонтичних дуг.

1.1. Задачі, що розв’язуються веб-системою

Задачами розробленої системи є:

- можливість завантаження 3D моделі у форматі STL;
- обрахунок мінімального циліндру, що обмежує задану модель за допомогою алгоритму Вельцля з лінійною складністю;
- візуалізація обрахованого мінімального циліндру, точок 3D моделі та виведення результатів на екран.

Кожна задача реалізується окремим мікросервісом, який передає і приймає дані з інших мікросервісів.

Сервіс для завантаження 3D моделі у форматі STL зчитує дані з файлу, зокрема координати вершин трикутних граней, з яких складається фігура і передає їх на вихід у форматі JSON.

Модуль обрахунку мінімального габаритного циліндру приймає на вхід хмару точок у форматі JSON та визначає мінімальну обмежуючу окружність для проекції

точок на площину XOY. Після чого знаходиться відстань між точками з найменшим і найбільшим значенням координати Z, яка буде визначати висоту циліндра. На вихід даний мікросервіс буде віддавати діаметр і висоту циліндра.

Сервіс візуалізації даних буде приймати на вхід як результати першого сервісу, зокрема хмару точок, так і результати другого сервісу – радіус і висоту мінімального циліндру, що описує цю хмару точок. Взаємодія цих підсистем і складає цілісну веб-систему (рисунок 1.1).

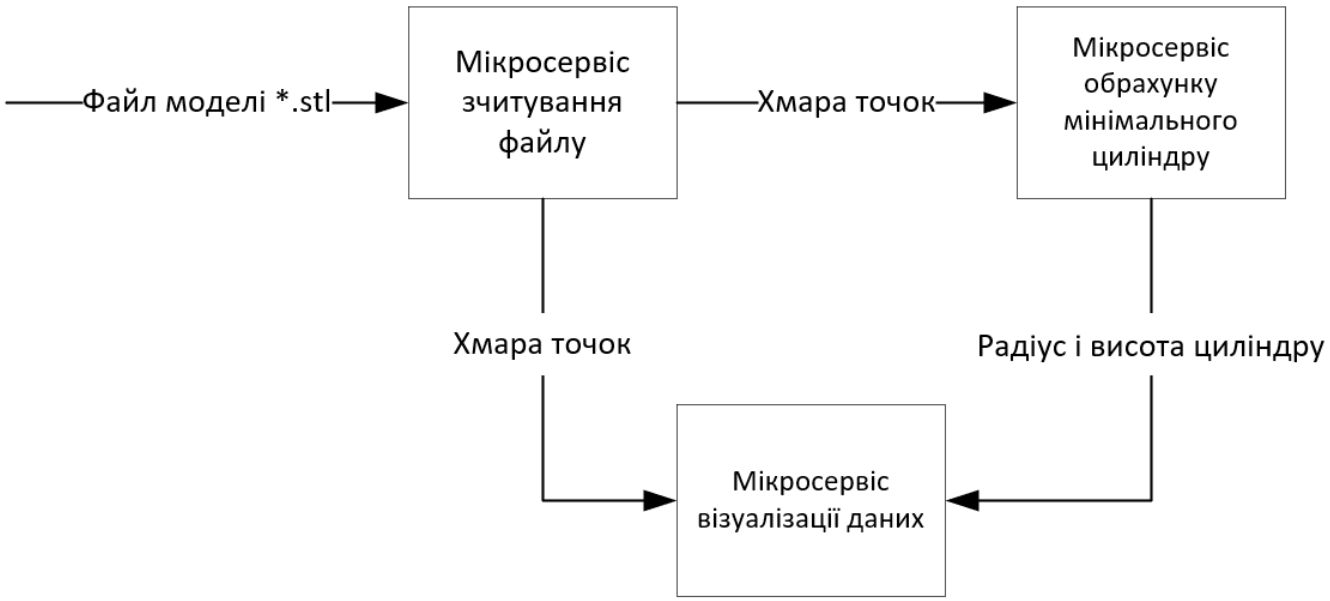

Рисунок 1.1 — Схема інформаційних потоків між підсистемами

Обмін даними між цими підсистемами відбувається за рахунок REST API (скорочення від англ. Representational State Transfer – “передача даних управління”) – архітектурний стиль взаємодії компонентів розподіленого додатка в мережі.

Мікросервіси написані за допомогою мови програмування JavaScript та середовища Node.js, яке дозволяє використовувати дану мову для написання серверної частини веб-системи.

Візуалізація даних відбувається за рахунок бібліотеки для роботи з тривимірною графікою Three.js.

1.2. Огляд існуючих програмних систем пошуку мінімального габаритного циліндру

Існує декілька програмних засобів вирішення задачі мінімального габаритного циліндру, але вони не є повноцінними програмними системами. Дані рішення представляють собою програмні модулі, реалізовані мовою програмування Python. Розглянемо один з них.

Дана реалізація представляє собою рішення задачі мінімального циліндру за допомогою модифікованого OBB (скорочення від англ. Oriented Bounding Box – “ орієнтований обмежуючий паралелепіпед ”) (рисунок 1.2).

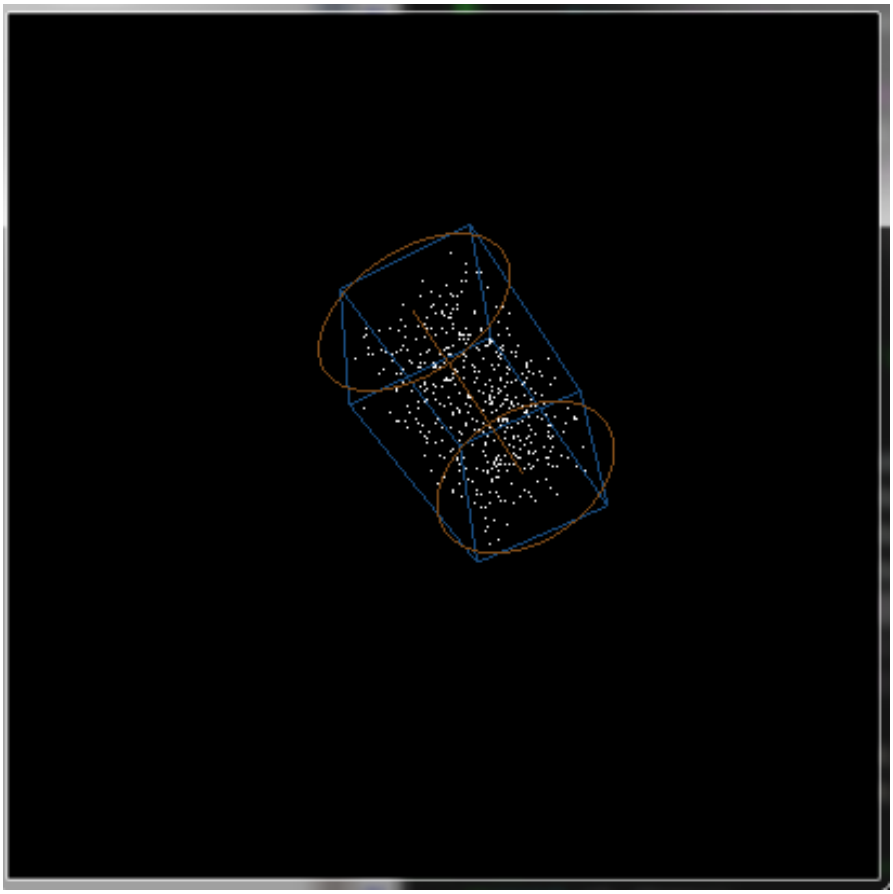

Рисунок 1.2 — Приклад роботи програмного модулю пошуку мінімального циліндру за допомогою орієнтованого обмежуючого паралелепіпеда

Серед недоліків даної реалізації можна виділити:

- дані представляють собою лише випадкові, а не завантажені точки певної 3D моделі;
- модуль не представлений у вигляді цілісної системи і його робота можлива лише при скачуванні на локальний комп’ютер з умовою встановленого Python.

Отже, можна зробити висновок, що дана програма є лише прикладом роботи алгоритму, а не повноцінною програмною системою.

Також можна виділити веб-реалізацію алгоритму мінімальної обмежуючої окружності, що лежить в основі розробленої системи пошуку мінімального габаритного циліндру (рисунок 1.3).

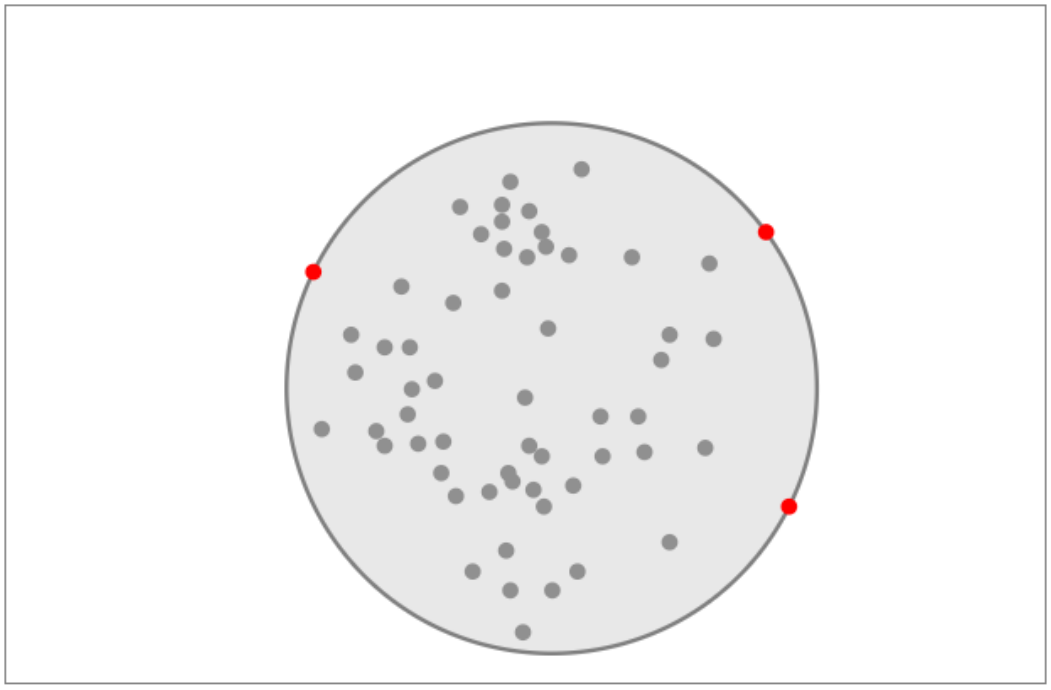

Рисунок 1.3 — Приклад роботи веб-реалізації алгоритму мінімальної обмежуючої окружності

Цей варіант системи також є лише прикладом роботи алгоритму з використанням випадкових точок [1].

Висновки до розділу 1

В розділі 1 було зазначено цільову аудиторію потенційних користувачів системи і основні задачі, які повинна виконувати система. Було розглянуто вхідні та вихідні дані. Окрім цього було побудовано схему взаємодії інформаційних потоків для наочності роботи системи. Також було проаналізовано існуючі програмні рішення, що включають в себе алгоритми розрахунку мінімальних обмежуючих тіл і фігур. Було розглянуто засоби реалізації і недоліки представлених рішень.

2. ЗАСОБИ РОЗРОБКИ

Для написання коду програмного продукту було використано мову програмування JavaScript.

Подібний вибір обумовлений зручністю мови JavaScript для створення веб-продуктів, в тому числі їх серверних компонентів.

Серверна сторона написана за допомогою середовища Node.js, яке перетворює мову програмування JavaScript у машинний код та дозволяє підключати звнішні бібліотеки, що написані на різних мовах і взяємодіяти з пристроями вводу-виводу за допомогою API (скорочення від англ. Application Programming Interface – “програмний інтерфейс додатків”).

Зображення даних відбувається за допомогою бібліотеки Three.js, що облегшує роботу з 3D графікою у браузері.

2.1. Середовище Node.js

Node.js – це асинхронне серверне оточення на JavaScript–рушієві Chrome V8, робота якого заснована на подіях. Дане середовище було спроектовано для побудови масштабованих мережевих додатків. Воно схожа за дизайном, та було створено під впливом таких систем як Event Machine в Ruby та Twisted у Python.

Середовище Node використовує подієву модель значно ширше, ніж аналоги, оскільки цикл подій (event loop) було прийнято за основу оточення замість того, щоб використовувати його в якості бібліотеки. В інших подібних системах завжди виконується блокування виклику, щоб запустити цикл подій. Зазвичай поведінка визначається через функції зворотного виклику на початку програмного коду, а в кінці запускається сервер через блокуючий (синхронний) виклик як EventMachine::run(). В Node немає нічого подібного до виклику початку циклу подій. Відбувається просте входження до подієвого циклу після запуску скрипту на виконання. Отже, система

Node виходить з подієвого циклу тоді, коли не залишається зареєстрованих функцій зворотного виклику. Така поведінка схожа на поведінку браузерного JavaScript: подієвий цикл прихований від користувача [2].

2.2. NPM: Менеджер пакетів Node і фреймворк Express

Якщо мова йдеться про середовище Node.js, неможливо не згадати про вбудовану в неї підтримку керування пакетами, для якої застосовується інструмент NPM, який за замовчуванням є в будь-якій встановленій версії платформи.

Для реалізації системи був використаний один з найпопулярніших сучасних NPM-модулей – фреймворк Express.

Express.js – фреймворк web-додатків для Node.js програмне забезпечення з відкритим вихідним кодом. Він спроектований для створення веб-додатків з широким набором функцій для створення мобільних, веб-додатків та API.

2.3. Мова програмування JavaScript

JavaScript (JS) — об'єктно-орієнтована, прототипна, динамічна мова програмування. Найширше вона використовується як мова сценаріїв для веб-сторінок, тобто надає можливість змінювати зовнішній вигляд та структуру веб-сторінки на стороні клієнта — пристрої користувача, асинхронно передавати дані на сервер та отримувати їх від нього. Окрім того вона може використовуватися в інших програмних продуктах – розглянутому вище Node.js або Apache CouchDB.

Мова програмування JavaScript характеризується як скриптова та прототипно-орієнтовна мова програмування з динамічною типізацією. Також JavaScript частково підтримує й інші парадигми програмування, такі як: функціональну та імперативну і деякі властивості архітектури як: слабку та динамічну типізацію, керування пам'яттю автоматично, функції як об'єкти першого класу та прототипне наслідування.

Стандартом мови JavaScript є ECMAScript – мова програмування, що використовується в якості основи для побудови інших скриптових мов, таких як: JavaScript, JScript, ActionScript.

Мова JavaScript находить застосування у:

- написанні сценаріїв (скриптів) сторінок для надання їм інтерактивності та динамічності;
- написанні коду на стороні сервера за допомогою середовища Node.js;
- створенні односторінкових веб-додатків — SPA (скорочення від англ. Single Page Application) за допомогою різноманітних фреймворків, таких як Angular, React тощо;
- створенні мобільних додатків (React Native, Ionic);
- написанні скриптів в прикладних ПЗ (наприклад, в програмних продуктах як Apache JMeter, Adobe Creative Suite).

Не зважаючи на схожість назв, мови Java та JavaScript є двома різними мовами. Мова JavaScript з’явилась пізніше, ніж Java і була названа таким чином через популярність мови Java, оскільки вона була поширена в той час. Вони мають різну семантику, хоч і мають схожі риси в правилах іменування та стандартних бібліотеках. Синтаксис цих мов був успадкований від C, але дизайн та семантика JavaScript є результатом впливу таких мов як Scheme та Self.

Структурно JavaScript можна представити у вигляді об'єднання трьох чітко помітних одна від одної частин:

- ядро (ECMAScript);
- об’єктна модель браузера (Browser Object Model або BOM);
- об’єктна модель документа (Document Object Model або DOM).

Якщо розглядати JavaScript в відмінних від браузера середовищах, то об'єктна модель браузера і об'єктна модель документа можуть не підтримуватися.

Об'єктну модель документа іноді розглядають як окрему від JavaScript сутність, що узгоджується з визначенням DOM як незалежного від мови інтерфейсу документа. На противагу цьому ряд авторів знаходить BOM і DOM тісно взаємопов'язаними [3].

2.4. Об’єктна модель браузера

Об'єктна модель браузера – браузер-специфічна частина мови, що є прошарком між об'єктною моделлю документа і ядром. Призначенням об'єктної моделі браузера є забезпечення взаємодії і управління вікнами браузера. Кожне окреме вікно браузера є об'єктом window, центральним об'єктом DOM. Об'єктна модель браузера на даний момент не стандартизована, однак специфікація знаходиться в розробці WHATWG і W3C.

Крім управління вікнами, в рамках об'єктної моделі браузера, браузерами зазвичай забезпечується підтримка наступних сутностей:

- управління фреймами;
- підтримка затримки в виконанні коду та зациклюванні з затримкою;
- системні діалоги;
- управління інформацією про браузер;
- управління адресою відкритої сторінки;
- управління інформацією про параметри монітора;
- обмежене управління історією перегляду сторінок;
- підтримка роботи з HTTP cookie.

2.5. Об’єктна модель документа

Об'єктна модель документа – це інтерфейс прикладного програмування для роботи з XML і HTML документами. Згідно DOM, документ (наприклад, веб-сторінка) може бути представлений у вигляді дерева об'єктів, що володіють рядом властивостей, які дозволяють робити з ним різні маніпуляції:

- створення і додавання вузлів;
- зміна вузлів;
- отримання вузлів;
- зміна зв’язків між вузлами;

– видалення вузлів.

Вузол може містити у собі вбудований вузол, елемент, текст або коментар.

2.6. Бібліотека Three.js

Three.js – кросбраузерна бібліотека JavaScript з відкритим вихідним кодом, що використовується для створення та відображення анімованої комп'ютерної 3D графіки при розробці веб-додатків. Three.js скрипти можуть використовуватися разом з елементом HTML5 CANVAS, SVG або WebGL.

Three.js дозволяє створювати прискорену на GPU (скорочення від англ. Graphics processing unit – "графічний процесор") 3D графіку як частину сайту без підключення будь-яких плагінів для браузера, використовуючи мову програмування JavaScript. Це можливо завдяки використанню технології WebGL.

2.7. Середовище розробки Visual Studio Code

Visual Studio Code – це функціональний редактор коду для розробки проектів з використанням таких мов як JavaScript, TypeScript, CSS та HTML з можливістю налагодження коду та вбудованим командним рядком. Дана середа розробки, створена компанією Microsoft. Вона безкоштовна та має відкритий вихідний код, а також доступна на платформах Windows, Linux та macOS.

Дане середовище розробки дозволяє встановлювати плагіни, створені спільнотою користувачів, що робить зручним використання різноманітних фреймворків для розробки додатків (рисунок 3.1).

Окрім того даний редактор коду візуально підсвічує синтаксис та надає можливість автодоповнення, що мінімізує час та полегшує процес розробки проектів.

Середовище надає можливість налаштувати кольорову тему, тому його зручно використовувати користувачам з будь-якими вподобаннями.

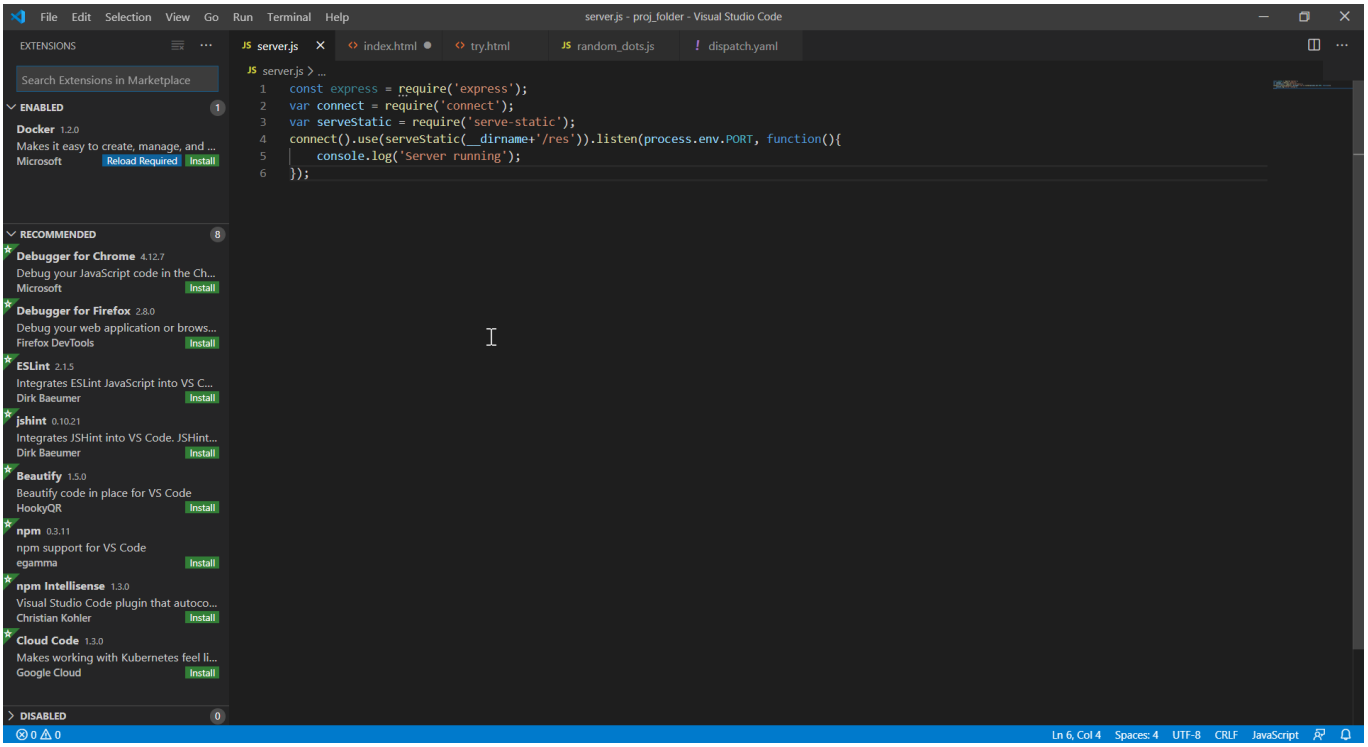

Рисунок 2.1 – Інтерфейс середовища Visual Studio Code

Отже, можна зробити висновок, що середовище розробки Visual Studio Code найкращім варіантом серед подібних редакторів коду, оскільки воно є потужним засобом для зручного написання коду з мінімальною витратою часу.

2.8. Хмарні технології

2.8.1. Огляд хмарних платформ

Для побудови веб-додатку було прийнято рішення використати хмарну платформу для збільшення ефективності розробки і розгортання системи. Хмарні платформи – це сервіси, що надають доступ за потребою до системних ресурсів, а саме сховища даних і обчислюваних потужностей. Ресурси оперативно надаються та звільняються з мінімальними експлуатаційними витратами чи зверненнями до провайдера.

Існує 3 стандартних сервісних моделі. Провайдери хмарних обчислень надають свої “послуги” відповідно до моделі:

- Інфраструктура як послуга (IaaS);
- Платформа як послуга (PaaS);
- Програмне забезпечення як послуга (SaaS);

IaaS (Infrastructure-as-a-Service) – ця модель характеризується наданням споживачу інформаційно-технологічних ресурсів, а саме віртуальних серверів з певними обсягами пам’яті та обчислювальною потужністю. Апаратним забезпеченням займається провайдер. На віртуальні сервери провайдером встановлюється ПО для створення віртуальних машин, а установкою і підтримкою власного ПЗ користувач займається самостійно. Під контроль провайдера підпадає тільки фізична та віртуальна інфраструктура. Прикладами хмарних провайдерів з сервісною моделлю IaaS є: Microsoft Azure, Amazon EC2, GigaCloud, Hetzner Cloud. Клієнтами IaaS є системні адміністратори компаній.

PaaS (Platform-as-a-Service) – сервісна модель, в якій хмарний провайдер надає доступ до операційних систем, систем управління базами даних, засобам розробки і тестування. На відміну від IaaS провайдер контролює не тільки сервери, системи зберігання даних і обчислювальні потужності, але й пропонує користувачеві на вибір певні платформи і засоби управління ними. Приклади PaaS: Google App Engine, Microsoft Azure, IBM Bluemix, VMWare Cloud Foundry. Користувачами PaaS-сервісів є розробники ПЗ.

SaaS (Software-as-a-Service) – є найпоширенішою хмарною моделлю. Програми та сервіси розробляються і обслуговуються провайдером. Він розміщує їх в хмарі і пропонує кінцевому користувачеві через браузер або додаток на його ПК. Оновленням і технічною підтримкою програм займається провайдер. Клієнти лише вносять абонплату (або користуються сервісом безкоштовно). SaaS-сервіси можуть надавати місце для зберігання файлів (Dropbox), офісний пакет документів для роботи (Google Doc, Microsoft Office 365), допомагати організовувати фотографії (Flickr) або спілкуватися з іншими людьми (Facebook). Основний клієнт SaaS-сервісів – звичайний користувач [4].

На наступному малюнку зображено моделі хмарних сервісів, що організовані у вигляді шарів піраміди з прикладами сервісів та їх цільовою аудиторією (рисунок 2.2).

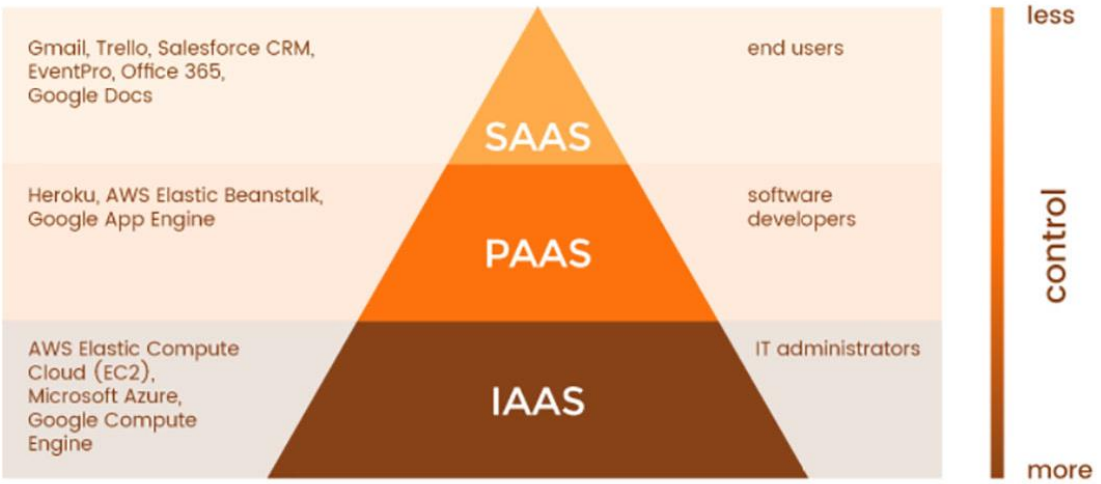

Рисунок 2.2 – Хмарна піраміда IaaS, PaaS, SaaS

2.8.2. Вибір платформи хмарних обчислень

Серед існуючих платформ є декілька лідерів хмарних обчислень, а саме:

- Google Cloud (рис. 2.3);
- Microsoft Azure (рис. 2.4);

Обидві ці платформи мають як PaaS так і IaaS сервіси.

На основі розглянутих моделей можна зробити висновок, що для реалізації поставленої задачі доцільним буде використовувати хмарний сервіс типу PaaS, оскільки він призначений саме для поодиноких розробників програмного забезпечення.

Головною перевагою платформи Google App Engine є відсутність потреби самостійного адміністрування віртуальних машин. Microsoft Azure має схожі характеристики, але більший поріг входження.

Платформа Google Cloud налічує багато сервісів, що використовуються для комфортної розробки програмного забезпечення будь-якої складності (рисунок 2.5).

Недоліками даних платформ є їх ціна, оскільки вони не є безкоштовними. Для ознайомлення обидві платформи надають можливість використання їх сервісів на 1 рік та певну суму на цей період.

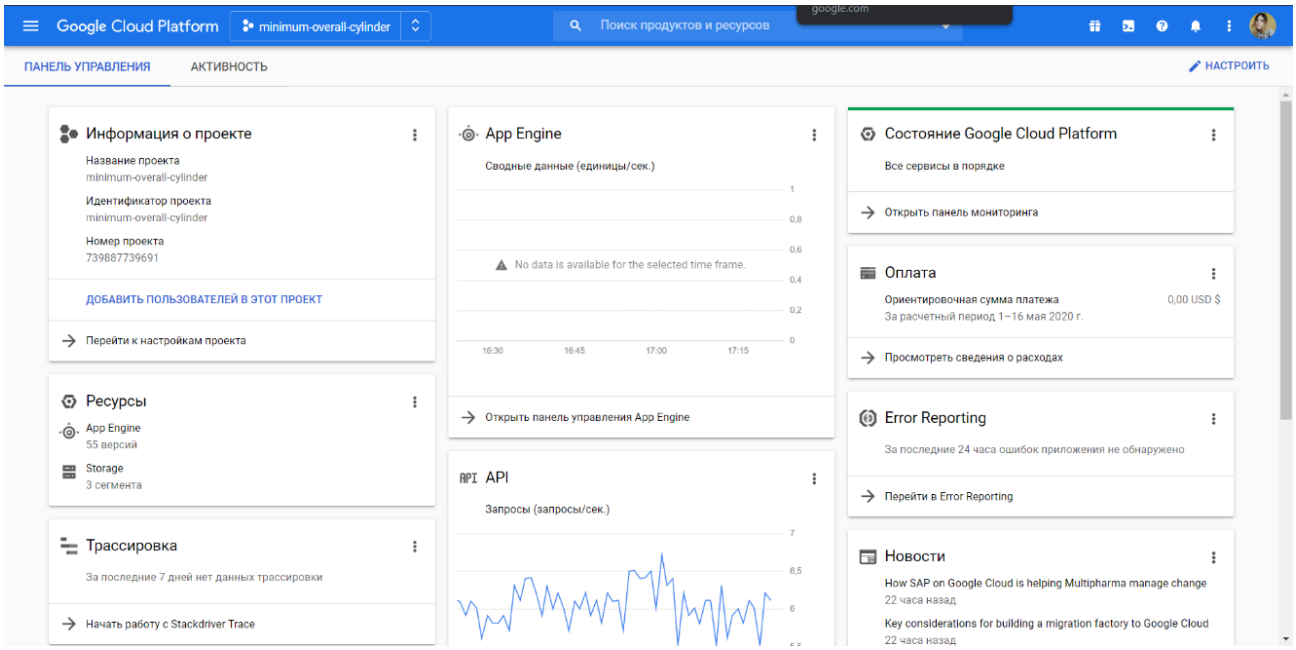

Рисунок 2.3 – Палень управління проектом у хмарній платформі Google Cloud

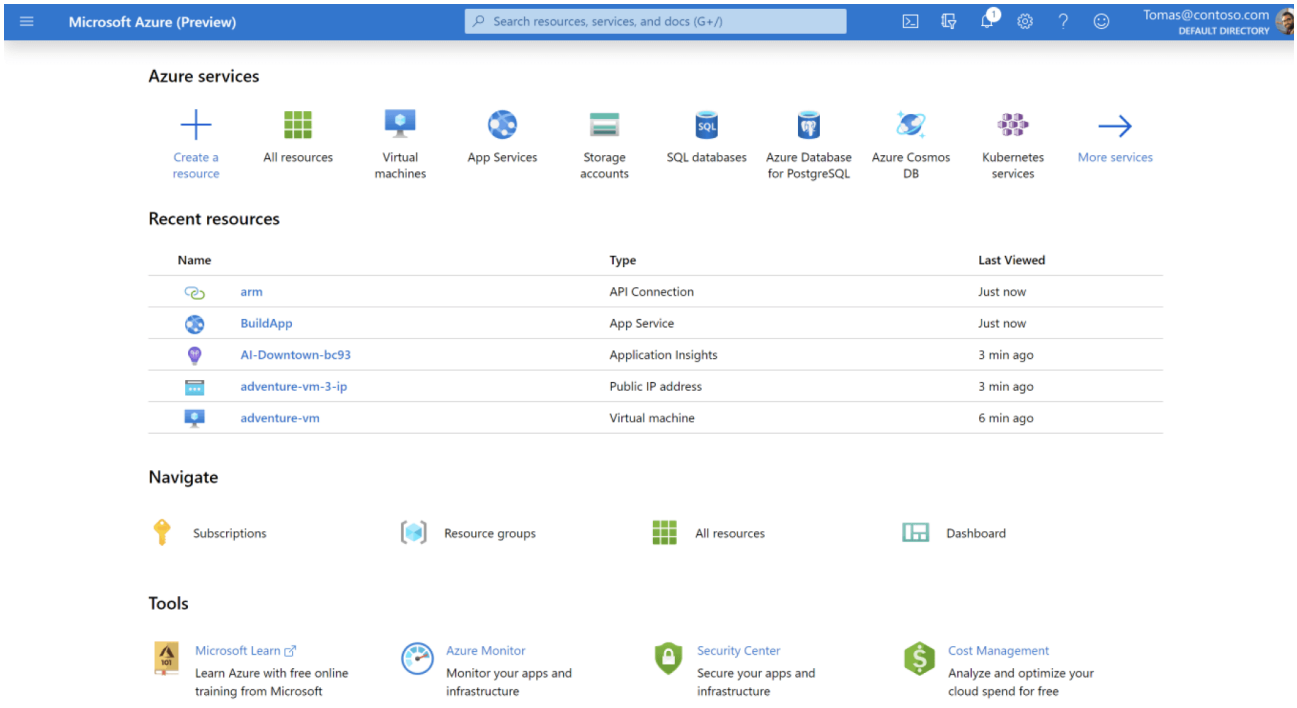

Рисунок 2.4 – Головна сторінка хмарної платформи Microsoft Azure

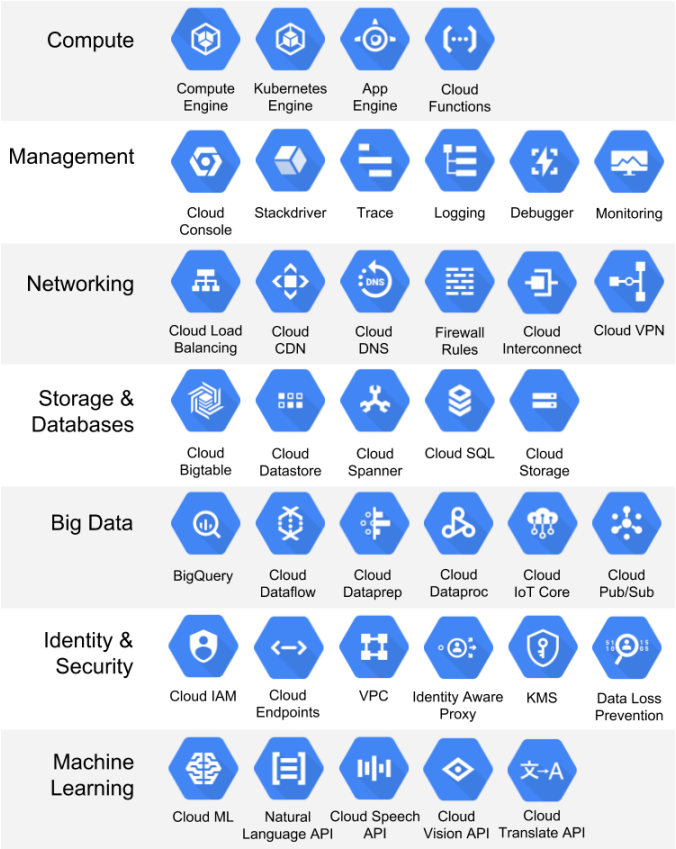

Рисунок 2.5 – Основні сервіси хмарної платформи Google Cloud

Умовами виходу з пробного періоду є закінчення пробного терміну або нестача грошей, що були видані на цей термін.

Зважаючи на це, було прийняте рішення обрати провайдером хмарних сервісів платформу Google Cloud, адже вона є більш зручною для розробки веб-системи невеликого масштабу і має більш вигідні умови пробного періоду.

2.10. Огляд мікросервісної архітектури побудови веб-додатку

Зважаючи на вибір хмарних технологій, на основі яких розробляється додаток для досягнення поставленої мети найбільше підходить мікросервісна архітектура. Вона відноситься до різновиду сервіс-орієнтовної архітектури, яка використовує модульний підхід до розробки програмного забезпечення. Він заснований на використанні розподілених і малозв’язних замінних компонентів, у випадку мікросервісної архітектури – мікросервісів.

Зручність використання мікросервісів полягає в простоті розгортання додатку на хмарній платформі. Кожного разу при зміні коду в певній частині програми необхідно розгортати цілий додаток, що може займати немало часу у випадку, якщо додаток має багато модулів. Окрім цього є шанс допустити помилку, що зробить всю систему непрацездатною. Це може спричинити ще більше сповільнення розробки. В свою чергу мікросервісна архітектура дозволяє розгортати окремі модулі у вигляді мікросервісів незалежно один від одного, що дозволяє пришвидшити розробку системи, оскільки окремі модулі легко протестувати на локальній машині та швидко розгорнути на сервері. За необхідністю модуль можна розширити або повністю замінити, не впливаючи на інші мікросервіси.

Окрім того перевагою мікросервісної архітектури є роздільність процесів, завдяки якій у кожного сервіса є своя обмежена кількість ресурсів, тому окремий модуль не може використати всю пам’ять або ЦПУ хосту.

Існує декілька різних шаблонів розгортання мікросервісів:

- Декілька екземплярів сервісів на хост (Multiple Service Instances per Host pattern);
- Екземпляр сервісу на кожен контейнер (Service Instance per Container Pattern);
- Екземпляр сервісу на віртуальну машину (Service Instance per Virtual Machine Pattern).

Кожен з цих шаблонів має свої переваги і недоліки.

Шаблон “Декілька екземплярів сервісів на хост” полягає в тому, що на декілька фізичних чи віртуальних хостів завантажується по декілька екземплярів сервісів. Перевагою даного шаблону є ефективне використання ресурсів, адже для кожного хосту сервіси користуються ресурсами однієї операційної системи. Ця особливість шаблону “Декілька екземплярів сервісів на хост” має і свій недолік: сервіси на одному хості не ізольовані один від одного і використовують один процес. Тому можливо лише відслідкувати використання ресурсів кожного екземпляру сервісу, але не обмежувати ресурси, що використовує екземпляр. Як наслідок один невірно

працюючий екземпляр мікросервісу може використовувати всю пам’ять та ЦПУ хосту.

В свою чергу за шаблоном “Екземпляр сервісу на кожен контейнер” кожен мікросервіс упаковується в окремий образ-контейнер. Кожен образ – це образ файлової системи, що складається з усіх додатків і бібліотек, які необхідні для запуску сервісу. Деякі образи більш легкі, а деякі складаються з повної кореневої файлової системи Linux. Перевагою даного шаблону розгортання мікросервісів є ізольованість кожного мікросервісу один від одного. Недоліком контейнерів є те, що вони менш безпечні, оскільки вони спільно використовують ядро операційної системи хосту. Окрім цього необхідно самостійно адмініструвати образи контейнерів у випадку, якщо не використовується розміщене контейнерне рішення, наприклад Google Container Engine або Amazon EC2 Container Service (ECS).

Для реалізації веб системи було обрано шаблон “Екземпляр сервісу на віртуальну машину”. Він полягає в пакуванні кожного сервісу як образ віртуальної машини (VM) (рисунок 2.6).

Головною перевагою такого підходу є повна ізоляція сервісів один від одного, кожен з яких має фіксований об’єм ресурсів процесора і пам’яті та не може використовувати ресурси інших сервісів. Для даного шаблону хмарна інфраструктура Google Cloud надає корисні функції, такі як балансування навантаження і автоматичне масштабування.

Ще одною перевагою даного паттерну є те, що віртуальна машина включає в себе технології реалізації сервісу, отже після того, як сервіс упакований у віртуальну машину, він стає чорною скринькою. API управління віртуальною машиною стає API надання послуг. Розгортання стає набагато простіше і надійніше.

Недоліком використання цього шаблону є менш ефективне використання ресурсів, оскільки кожен екземпляр сервісу включає свою операційну систему, на яку витрачається частина ресурсів. Окрім цього завантаження нової версії сервісу відбувається повільніше через необхідність створення віртуальної машини. Це відбувається через розмір віртуальних машин і час, необхідний на завантаження операційної системи. Але цей недолік відноситься не до всіх віртуальних машин [5].

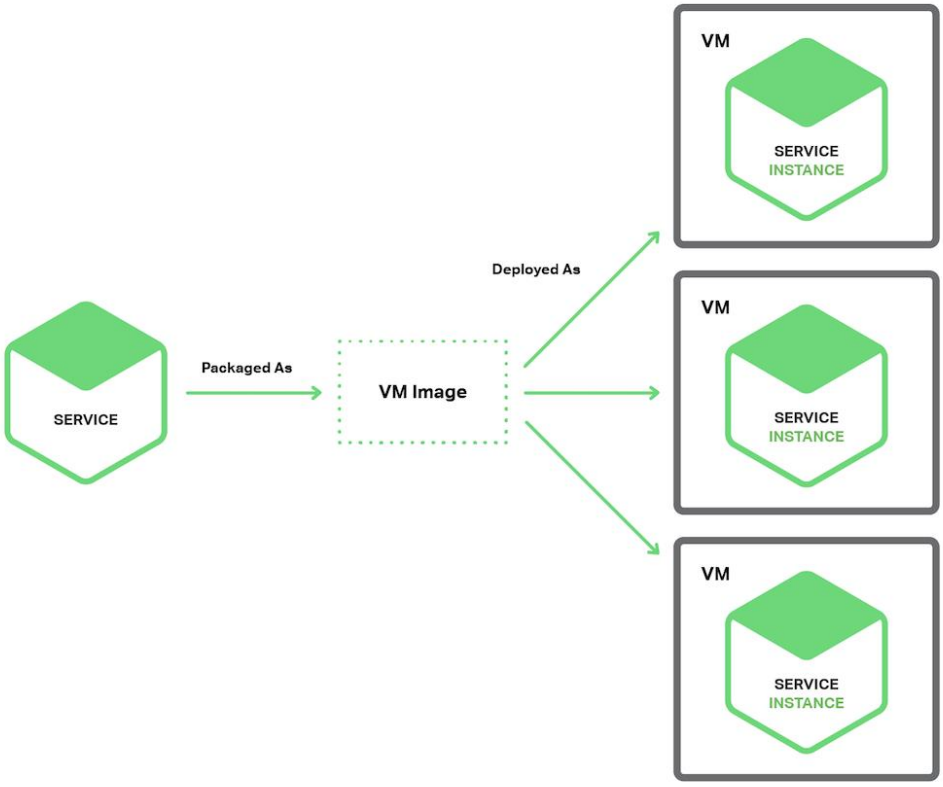

Рисунок 2.6 – Паттерн розгортання мікросервісів “Екземпляр сервісу на віртуальну машину”

Не дивлячись на всі недоліки, даний шаблон використовується найширше комерційними компаніями через його надійність та простоту розгортання і адміністрування. Саме тому він був обраний для реалізації веб-системи.

Висновки до розділу 2

Цей розділ описує основні засоби, що були використані при написанні програмного забезпечення. В ньому наведено середовище розробки, використана бібліотека та середовище, в якому був написаний програмний код.

Окрім цього в розділі було розглянуто і проаналізовано моделі хмарних обчислень. Було розглянуто платформи на основі обраної моделі, зазначені їх переваги і недоліки та обґрунтований вибір платформи для розробки і розміщення

додатку. Також було розглянуто і проаналізовано шаблони розгортання мікросервісної архітектури на хмарній платформі.

3. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

В цьому розділі наведено функціональність системи, обґрунтування і аналіз обраного алгоритму і спроектовану архітектуру системи розрахунку мінімального габаритного циліндру 3D моделі.

3.1. Функціональність системи

Для опису функціональності розробленої системи було побудовано діаграму прецедентів (рисунок 3.1).

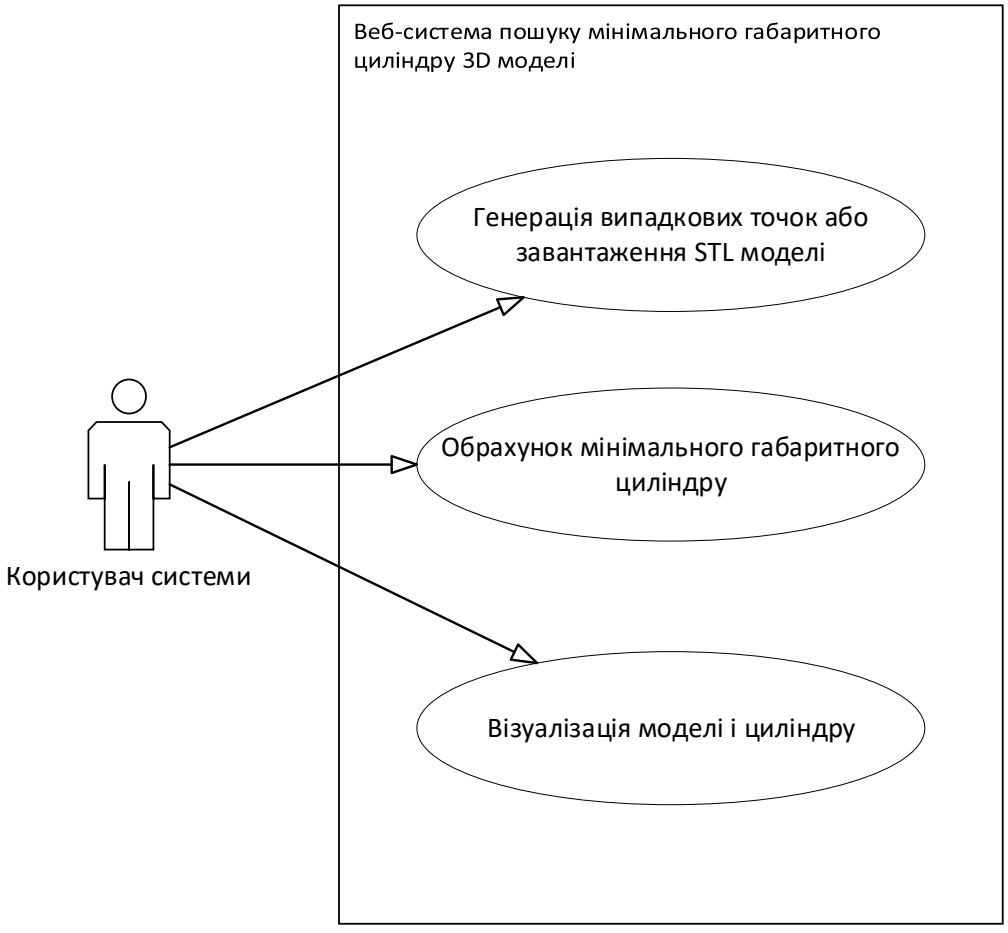

Рисунок 3.1 – Діаграма прецедентів з варіантами використання системи

На діаграмі прецедентів зображуються діючи особи (актори) і варіанти використання (прецеденти). Зокрема для розробленої системи актором є користувач системи, а варіантами використання – результат взаємодії користувача з мікросервісами додатку.

3.2. Алгоритм пошуку мінімального габаритного циліндру

Для вирішення задачі мінімального габаритного циліндру немає готового алгоритму, тому перш за все необхідно вирішити, що представляє собою мінімальний циліндр. В для досягнення мети роботи доцільним критерієм мінімальності буде радіус циліндра. В цьому випадку задача переходить до пошуку мінімальної обмежуючої окружності.

Існує декілька алгоритмів для вирішення задачі про мінімальну обмежуючу окружність. Ось деякі з них:

- наївний алгоритм;
- алгоритм Христала та Пірса;
- алгоритм Елзінга та Хірна;
- алгоритм Шамоса і Хауі;
- алгоритм Вельцля.

Всі ці алгоритми головним чином відрізняються складністю. Тому доречно буде обирати алгоритм саме за цим критерієм.

Найгіршим варіантом є наївний алгоритм, який має складність $O(n^4)$. Він вирішує поставлену задачу шляхом перевірки окружностей, що задаються всіма парами та тройками точок.

Алгоритм Христала та Пірса має складність $O(nh)$, Елзінга та Хірна в найгіршому випадку – $O(h^3 n)$, а складність алгоритму Шамоса і Хауді – $O(n \log n)$.

Серед них виділяється алгоритм Еммеріха Вельцля, який має лінійну складність $O(n)$. Саме тому він і буде основою для створення веб-системи.

Даний алгоритм є рандомізованим і рекурсивним, а в якості аргументів виступають дві множини точок *S* і *Q*. Алгоритм обраховує найменшу окружність, що обмежує об'єднання даних множин, якщо будь-яка точка множини *Q* є граничною точкою можливої обмежуючої окружності. Вихідна задача про найменшу обмежуючу окружність може бути вирішена починаючи з множини *S*, в яку входять всі надані точки і з порожньої множини *Q*. При рекурсивному виклику алгоритму, множина *Q* збільшується до того моменту, як в неї не попадуть всі граничні точки.

Алгоритм випадково обирає точки множини *S*, використовуючи множину *P* оброблених точок і мінімальну окружність, що обмежує об'єднання *P* і *Q*. На кожній ітерації алгоритм перевіряє чи належить обрана точка *r* цій окружності. Якщо не належить, алгоритм замінює обмежуючу окружність шляхом рекурсивного виклику алгоритму на множинах *P* і *Q*+*r*. В залежності від того, замінюється окружність чи ні, *r* включається в множину *P*. Обробка кожної точки, таким чином, полягає в перевірці, чи належить точка окружності, і можливому рекурсивному виклику алгоритму. Можна показати, що *i*-та точка, що оброблюється має ймовірність $O(i / n)$ рекурсивного виклику, а отже складність є лінійною.

Псевдокод реалізації алгоритму має наступний вигляд (рисунок 3.2) [6]:

```
SmallestEnclosingCircle(P)
 Ensure: B ← ∅
 for i = 1 to n
 |   if p_i ∉ EncBall(B) then
 |     B ← {p_i}
 |     for j = 1 to i − 1
 |     |   if p_j ∉ EncBall(B) then
 |     |     B ← {p_i, p_j}
 |     |     for k = 1 to j − 1
 |     |     |   if p_k ∉ EncBall(B) then
 |     |     |     B ← {p_i, p_j, p_k}
 |     |     |   end if
 |     |     end for
 |     |   end if
 |     end for
 |   end if
 end for
 return B
```

Рисунок 3.2 – Псевдокод алгоритму пошуку мінімальної обмежуючої окружності

Висота циліндру буде визначатися за мінімальним і максимальним значенням координати Z точок хмари.

За результатами роботи алгоритму Вельцля було отримано наступні дані (Таблиця 3.1):

Таблиця 3.1

Таблиця вибірок часу роботи алгоритму для кількості точок моделі

Кількість точок у хмарі	Час роботи алгоритму (мс)				
	Вибірка 1	Вибірка 2	Вибірка 3	Вибірка 4	Середнє значення по вибірках
100	1	0	3	2	1.5
1000	3	2	2	3	2.5
3000	7	10	25	10	13
5000	10	17	17	20	16
10000	37	31	19	28	28.75
50000	140	115	180	178	153.25
70000	297	122	125	260	201
90000	269	198	153	109	182.25
100000	365	529	1053	2018	991.25

За отриманими даними побудовано графік (рисунок 3.3):

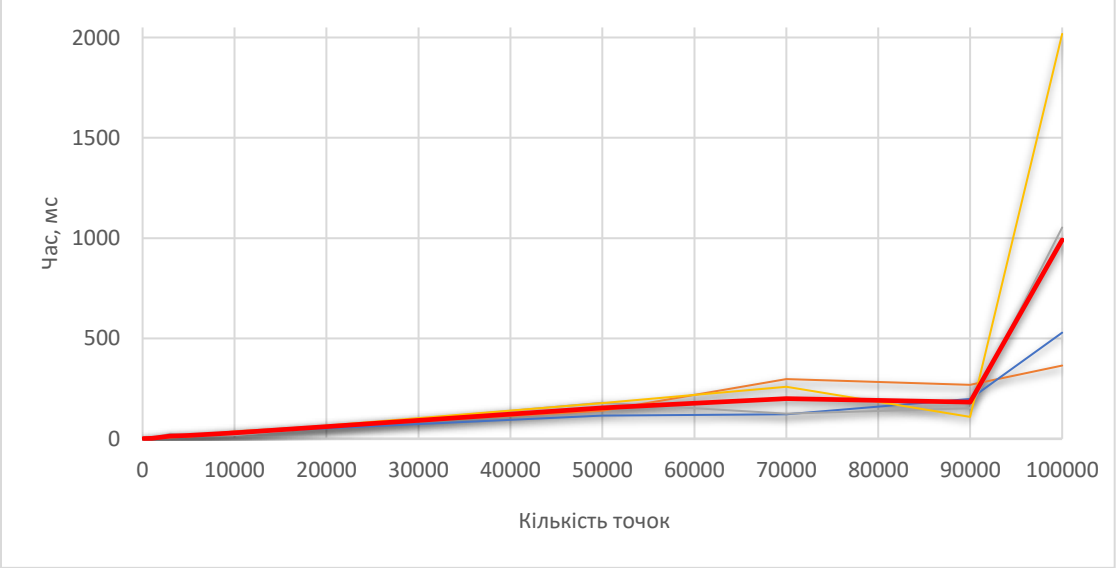

Рисунок 3.3 – Графік залежності часу роботи алгоритму від кількості точок в моделі (графік складності алгоритму)

На отриманому графіку кольоровими лініями позначені вибірки часу роботи алгоритму, а червоною – середнє значення по вибірках. Як видно з рисунку 3.3 залежність часу роботи алгоритму від розміру хмари точок має лінійний характер. Вище 90000 точок хмари відбувається різке збільшення часу роботи алгоритму. Час роботи алгоритму, що знаходиться в межах однієї секунди показує, що алгоритм може бути використаний в системах моделювання, що працюють в режимі реального часу.

3.3. Особливості реалізації мікросервісної архітектури

Для реалізації веб-системи був обраний мікросервісний архітектурний стиль, за яким застосунок будується як сукупність невеликих сервісів, кожен з яких працює у своєму власному процесі, а обмін даними відбувається за допомогою HTTP [4].

Для того, щоб мікросервіси були доступні всі одночасно та працювали коректно існує інструмент Docker. Цей інструмент використовують для пакування сервісу у образ (image) і віртуальні машини для одночасного запуску кількох мікросервісів.

Цей інструмент має ряд переваг та недоліків. Зокрема останні й найстабільніші версії Docker потребують вбудовану в операційну систему віртуальну машину, що робить неможливим його використання на деяких ОС. Окрім того система з використанням Docker за замовчуванням не розміщується на хостингу, отже вона буде доступна лише локально.

Перевагою Docker є його безкоштовне використання та відкритий вихідний код.

В свою чергу для перекриття цих недоліків використовується хмарна платформа Google Cloud. Вона має вбудований Docker для контейнеризації сервісів та розміщує додаток на хостингу Google App Engine, отже він буде доступний з будь-якої точки світу незалежно від операційної системи.

Для використання платформи необхідно зареєструватися в системі, створити новий проект і встановити на машину наступне програмне забезпечення:

- Python 2.7;

– Google Cloud SDK.

За умовою встановлення цих інструментів додаток можна розмістити на платформі за допомогою стандартного командного рядка (для ОС Windows це Cmd.exe).

Спочатку необхідно виконати команду “gcloud init” для авторизації на платформі.

Після створення головної директорії додатку необхідно зайти в неї через командний рядок та виконати команду “gcloud app create”. Після цього треба ініціювати створення додатку Node.js командою “npm init” і задати у файлі package.json в якості стартового скрипту файл серверу додатку і вказати версію середовища Node.js. Після цього створити файл app.yaml, який слугує точкою входу в додаток (рисунок 3.3).

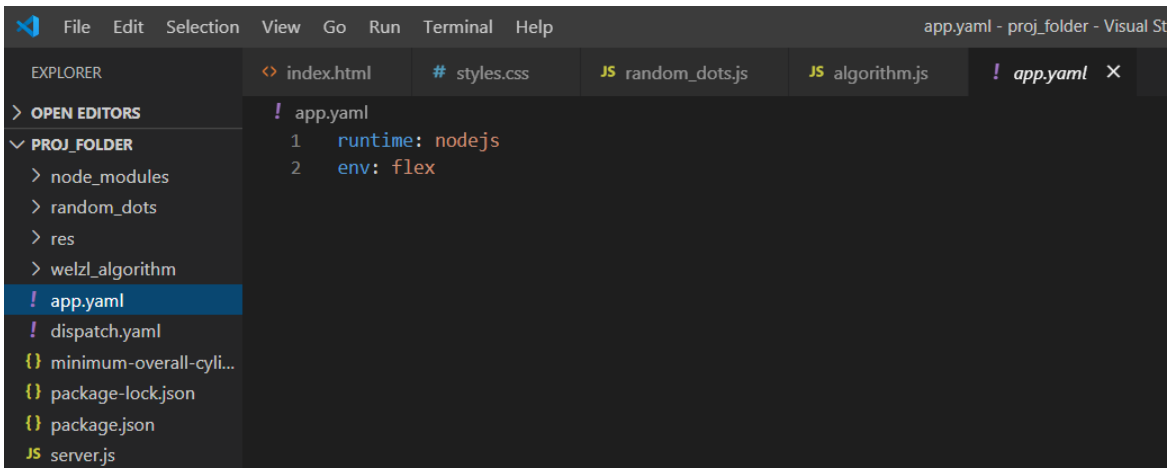

Рисунок 3.3 – Файл app.yaml, що слугує точкою входу в додаток

Нижче наведено приклад коду серверу, що завантажує клієнтську частину системи (рисунок 3.4).

Визначення порту як “process.env.PORT” дозволяє оточенню платформи самостійно обрати порт, який сервер повинен слухати для вхідних запитів. За замовчуванням Google App Engine використовує 8080 порт.

Для того, щоб розгорнути додаток, у командному рядку необхідно виконати комаду “gcloud app deploy” у головній директорії додатку (рисунок 3.5).

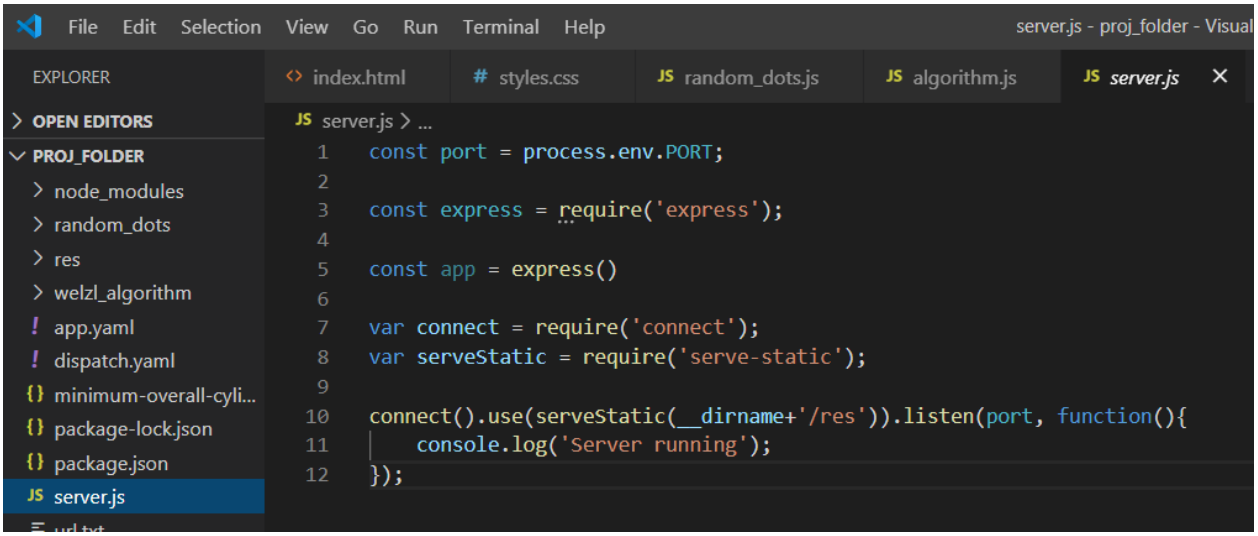

Рисунок 3.4 – Файл server.js, що завантажує клієнтську частину системи на хост

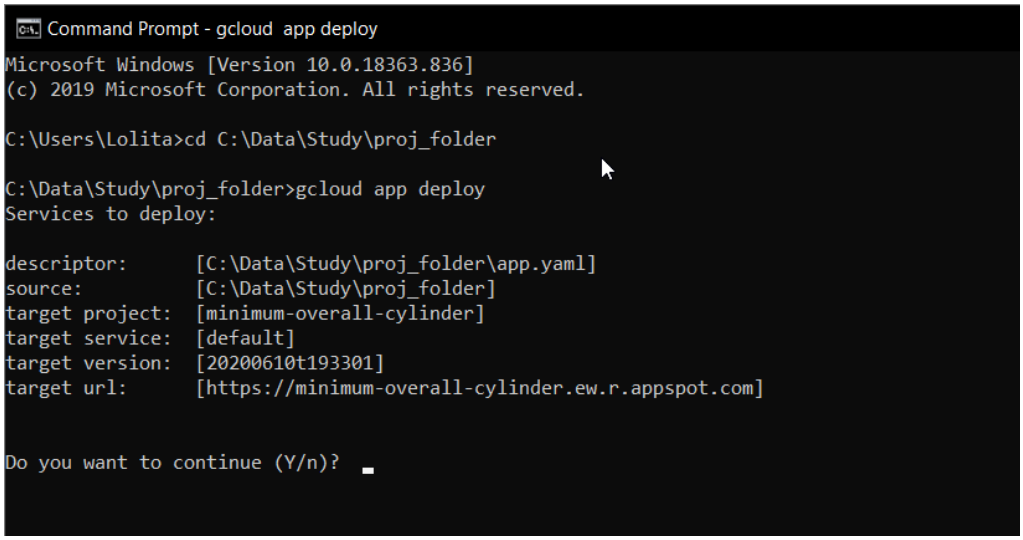

Рисунок 3.5 – Команда, що виконує розгортання додатку

Для додавання у проект мікросервісу створюється новий проект Node.js у головній директорії. Таким самим чином необхідно налаштувати package.json і створити файл [назва мікросервісу].yaml, в якому необхідно задати ім’я мікросервісу (рисунок 3.6).

Для того, щоб пов’язати сервіси між собою в головній директорії проекту створюється файл dispatch.yaml, в якому вказується назва мікросервісу і URL (скорочення від англ. Uniform Resource Locator – “уніфікований вказівник ресурсу”), за яким буде відбуватися доступ до сервісу (рисунок 3.7).

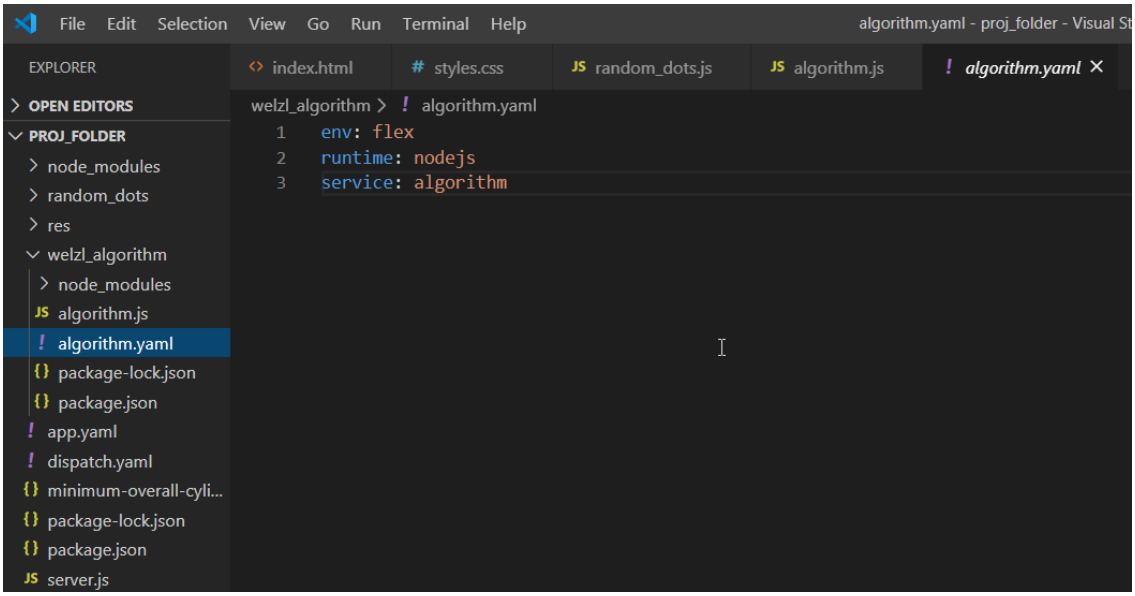

Рисунок 3.6 – Файл мікросервісу algorithm – algorithm.yaml

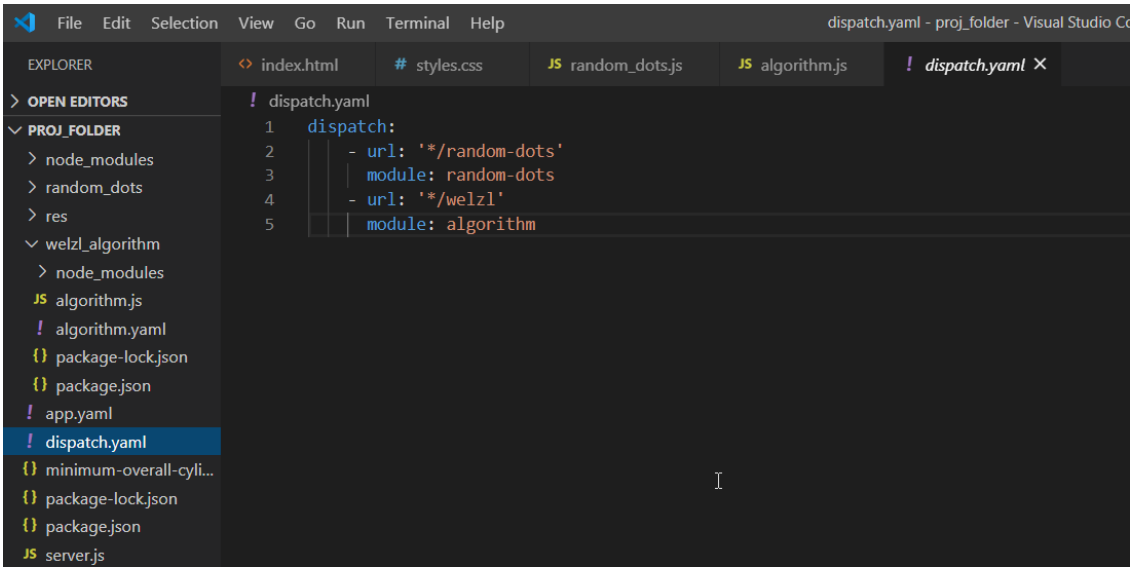

Рисунок 3.7 – Файл головної директорії додатку dispatch.yaml з посиланнями на мікросервіси і їх назвами

Після виконання цих дій для розгортання мікросервісу у командному рядку виконується команда “cloud app deploy [директорія мікросервісу]/[назва мікросервісу].yaml dispatch.yaml”.

3.4. Архітектура веб-системи

За обраним шляхом реалізації було створено схему архітектури системи розрахунку мінімального габаритного циліндру (рисунок 3.8).

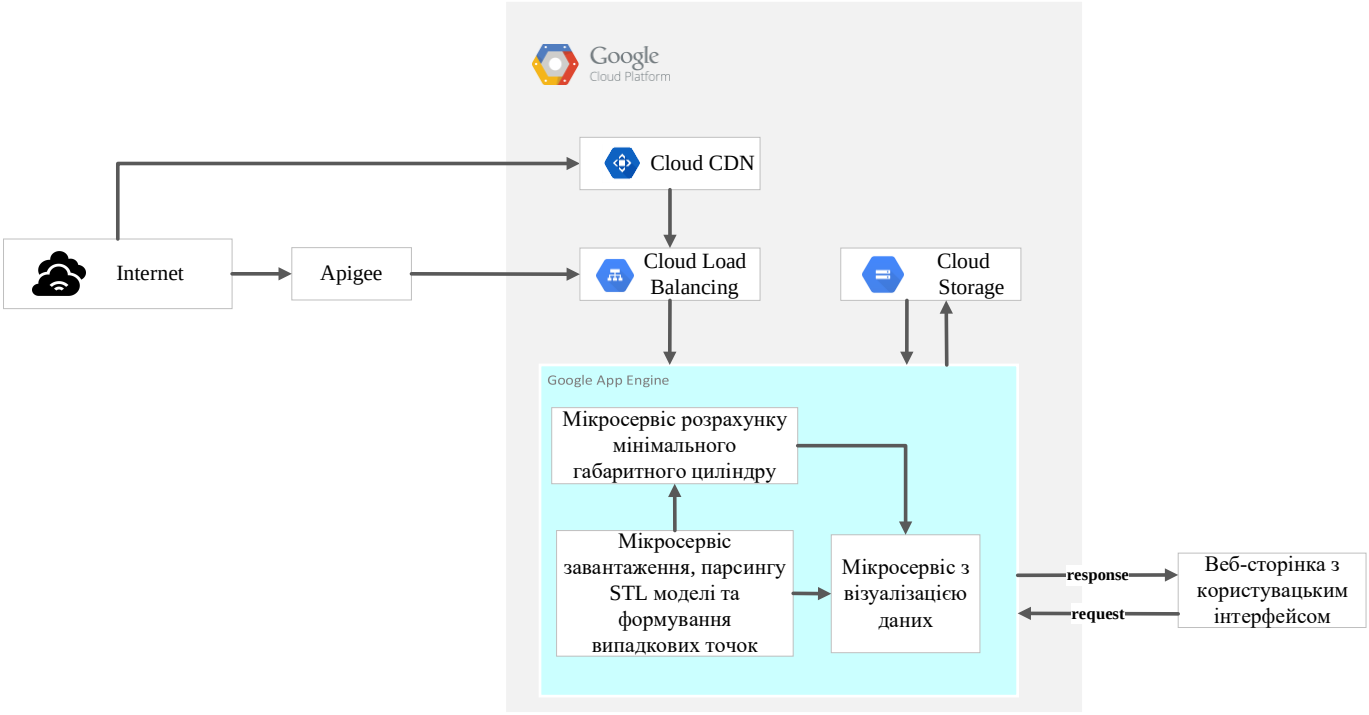

Рисунок 3.8 – Архітектура веб-системи на базі мікросервісів і хмарної платформи Google Cloud

Архітектура веб-системи заснована на використанні сервісів хмарної платформи Google Cloud, таких як

- Cloud CDN;
- Cloud Load Balancing;
- Cloud Storage.

Сервіс Google Cloud CDN забезпечує оперативну доставку контенту додатку користувачам, зважаючи на їх регіон.

Cloud Load Balancing автоматично розподіляє обчислювальні ресурси додатку і масштабує його зважаючи на кількість користувачів так кількість запитів на сервер в секунду.

Сервіс для зберігання даних Cloud Storage дозволяє завантажувати та використовувати файли у додатку.

Платформа Apigee відповідає за керування API і аналітикою.

Мікросервіси і клієнтська частина системи обмінюються даними за допомогою HTTP повідомлень request-response.

Висновки до розділу 3

В даному розділі було розглянуто функціональність системи, описано метод створення додатків за допомогою мікросервісної архітектури та розглянуто архітектуру веб-додатку розрахунку мінімального циліндру, що обмежує хмару точок.

4. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Цей розділ описує системні вимоги для оптимальної роботи веб-системи та сценарії роботи користувача з ним.

4.1. Системні вимоги до програмного продукту

Програмний продукт було розроблено та протестовано на комп’ютері Xiaomi Mi Notebook Air 13.3". Він має наступні технічні характеристики: чотирьохядерний процесор Intel ® Core i5 (1.6 - 1.8 ГГц), об’єм оперативної пам’яті – 8 ГБ, диск SSD на 256 ГБ та відеокарта GeForce MX150.

На інших пристроях тестування програмного продукту проведено не було, проте, програмний продукт може працювати і на комп’ютерах з гіршими технічними характеристиками.

Якщо брати до уваги те, що програмний продукт – це веб-сервіс, можна стверджувати, що система також коректно працюватиме на комп’ютерах з операційною системою Linux та macOS, а також на мобільних пристроях з операційною системою Android або iOS. Перегляд програмного продукту на різних пристроях можливий за допомогою веб-браузерів, таких як Google Chrome, Mozilla Firefox, Opera, Internet Explorer, Safari та інших.

Проте рекомендовано встановлювати останні версії вище зазначених браузерів, оскільки деякі функції програмної системи можуть не працювати на старих версіях.

4.2. Сценарій роботи користувача з програмою

Розроблена система є веб-орієнтовною, отже вона може бути доступна без попереднього встановлення для всіх платформ, на яких можуть бути встановлені

браузери. Програма доступна в середовищі мережі Інтернет. Головне вікно програми має наступний вигляд (рисунок 4.1):

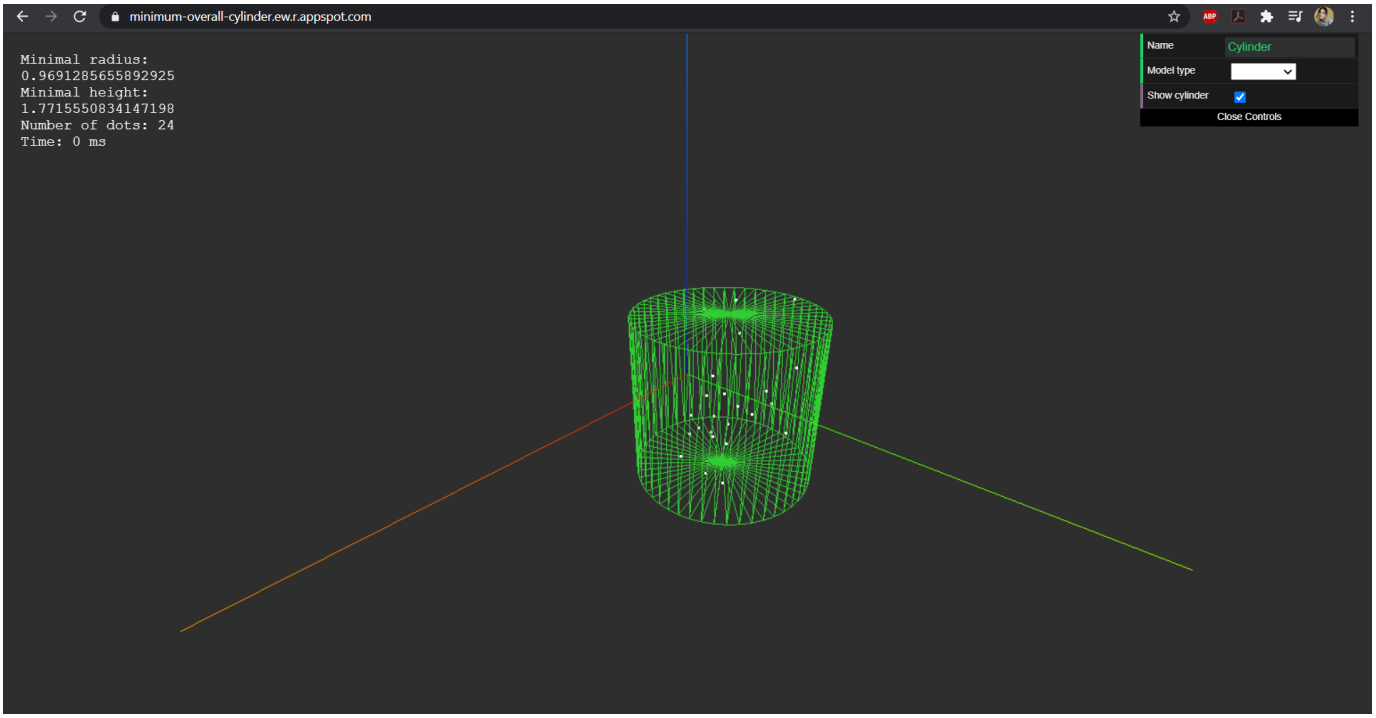

Рисунок 4.1 – Головне вікно веб-сервісу

Більша частина головного вікна зайнята областю, на якій відображається хмара точок і циліндр, що описує їх. При загрузці веб-сервісу за замовчуванням відображається хмара з випадковою кількістю випадкових точок.

За допомогою користувацького інтерфейсу, що розташований у верхній правій частині вікна сервісу можна вимкнути відображення циліндру (рисунок 4.2).

В програмі є можливість обрати джерело завантаження точок: з файлу формату STL, або випадково згенеровані точки (рисунок 4.3).

При виборі завантаження точок з файлу, на екрані відображається форма, завдяки якій можна завантажити файл формату STL з локальної директорії комп’ютера у систему (рисунок 4.4).

Після завантаження файлу та натискання кнопки “Upload” на екрані відображаються вершини трикутників (полігонів), з яких складається 3D модель (рисунок 4.5).

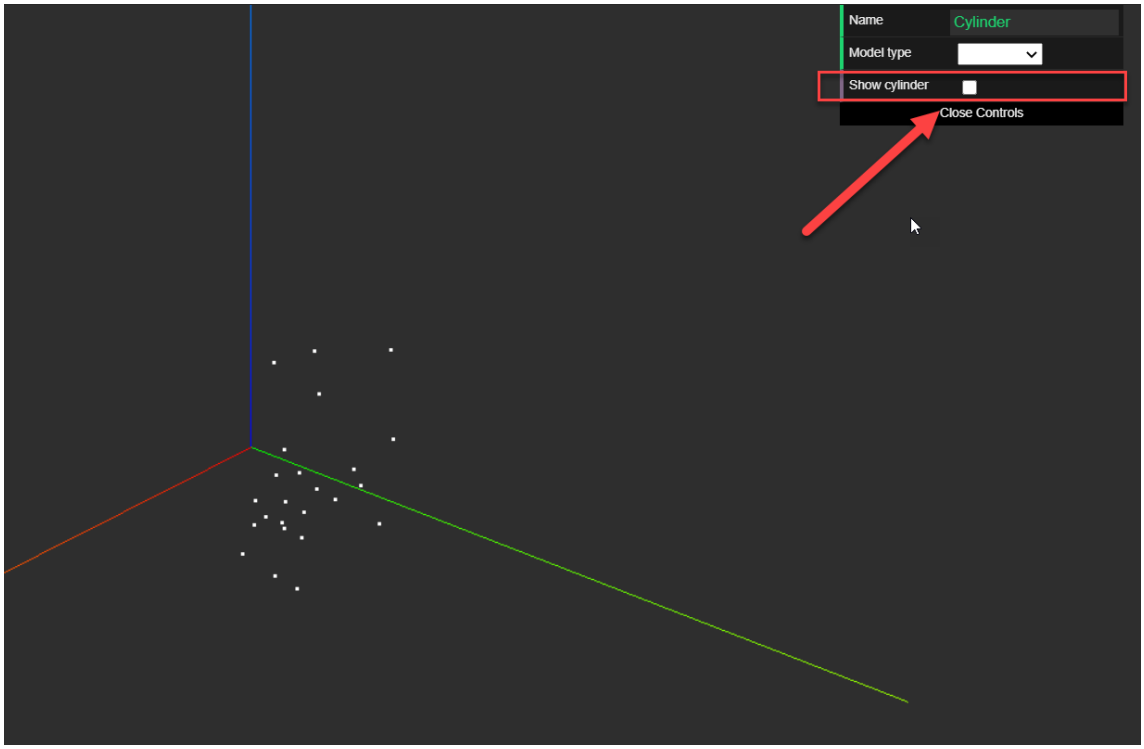

Рисунок 4.2 – Елемент інтерфейсу, що відповідає за вимкнення відображення циліндру

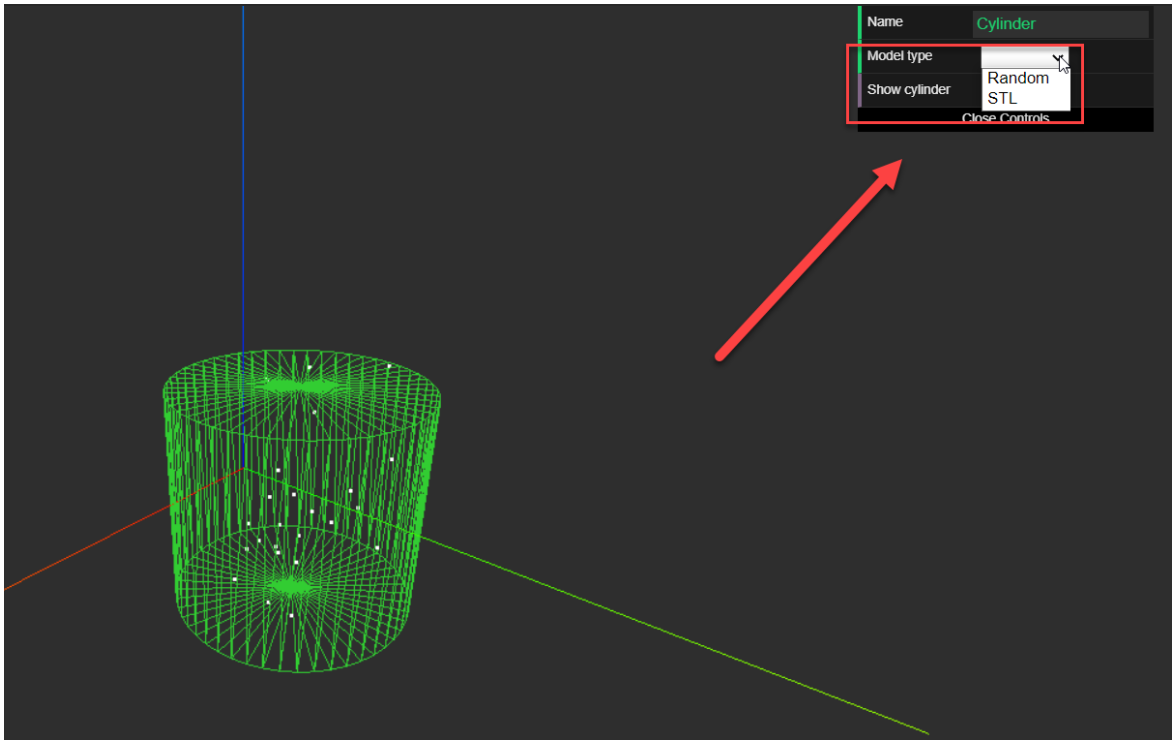

Рисунок 4.3 – Розкривний список з варіантами вибору джерела хмари точок

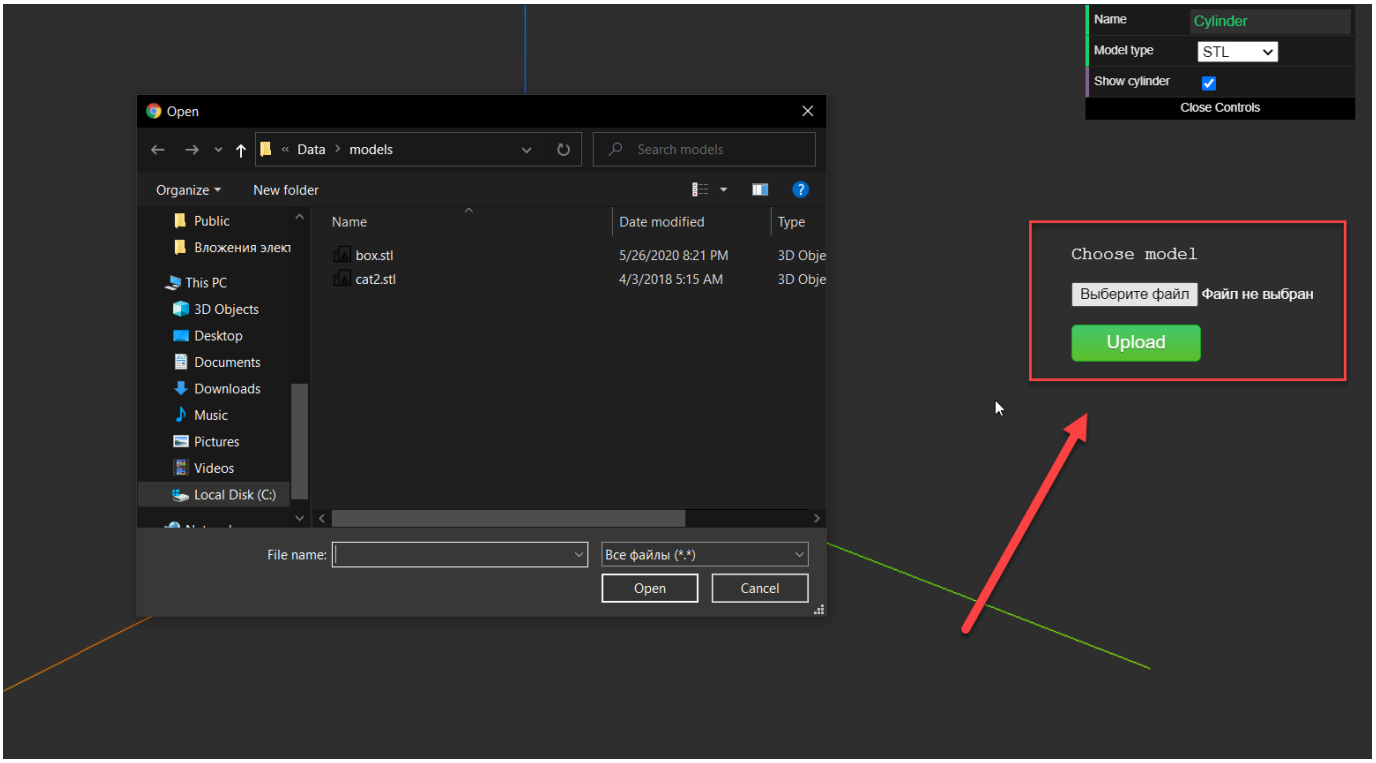

Рисунок 4.4 – Форма завантаження файлу формату STL

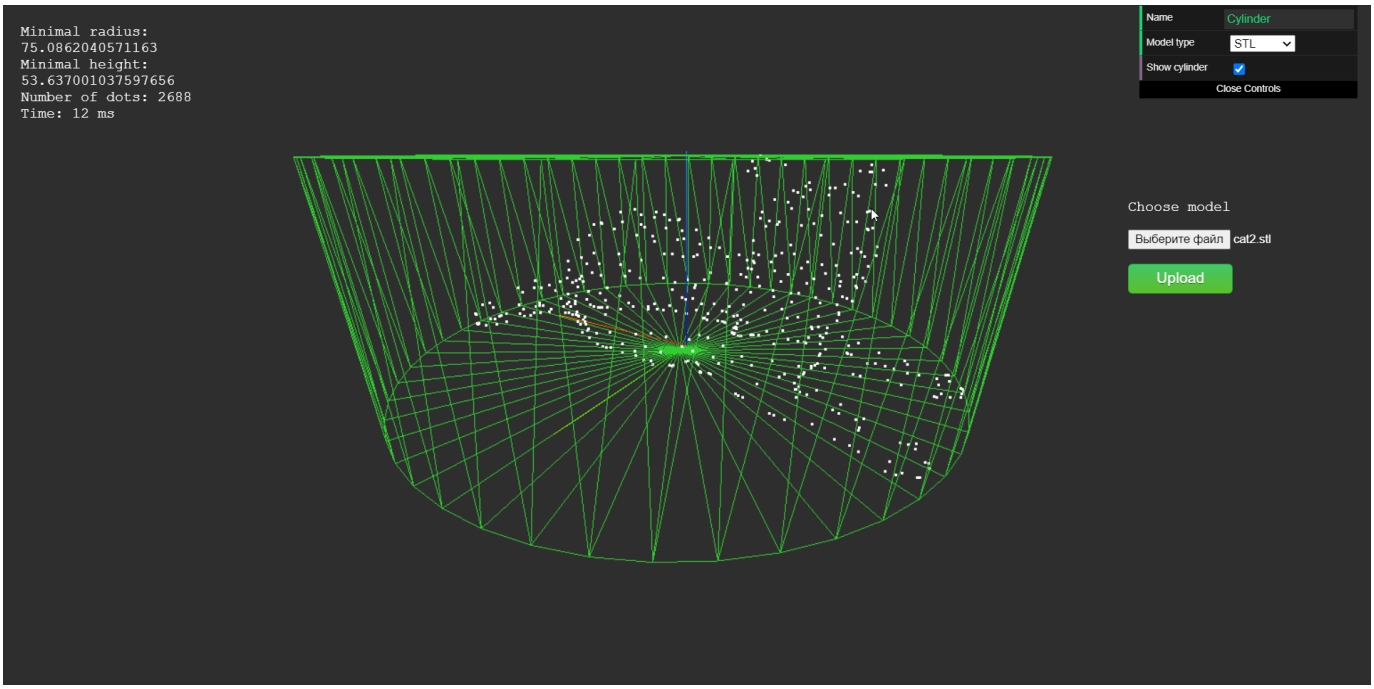

Рисунок 4.5 – Хмара точок завантаженої 3D моделі

При виборі джерелом завантаження генератор випадкових точок на екрані відображається форма, за допомогою якої можна ввести кількість точок, яку необхідно згенерувати (рисунок 4.6).

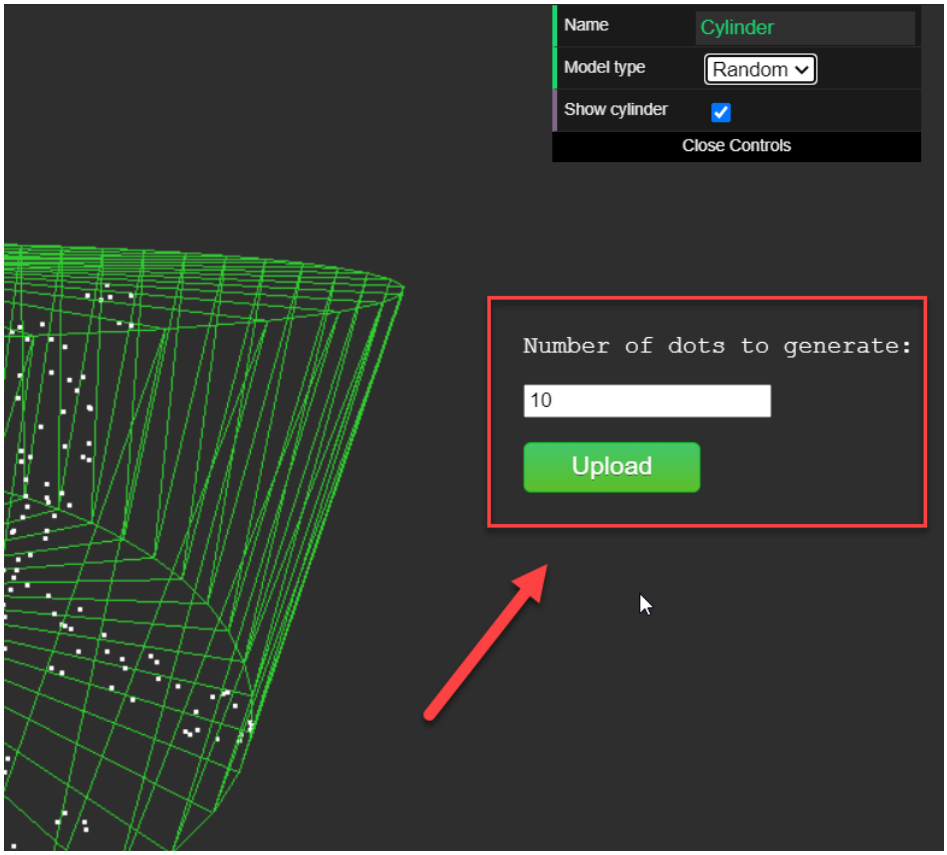

Рисунок 4.6 – Форма вводу кількості точок для генерації

У лівій верхній частині екрану знаходиться блок з результатами роботи алгоритму: мінімальним радіусом і висотою. Окрім цього відображається кількість точок та час роботи алгоритму у мілісекундах (рисунок 4.7).

Завантажену модель разом з циліндром можна роздивитися детальніше обертаючи, наближаючи чи віддаляючи камеру за допомогою комп'ютерної мишки (рисунок 4.8).

За вищенаведеними сценаріями роботи можна зробити висновок, що користувацький інтерфейс веб-системи є доволі простим і зручним у користування навіть для користувачів почтакового рівня.

Таким чином користувачами розробленої системи можуть бути як підприємці, так і випадкові користувачі мережі Інтернет, метю яких є знайти мінімальний циліндр, покриваючий хмару точок.

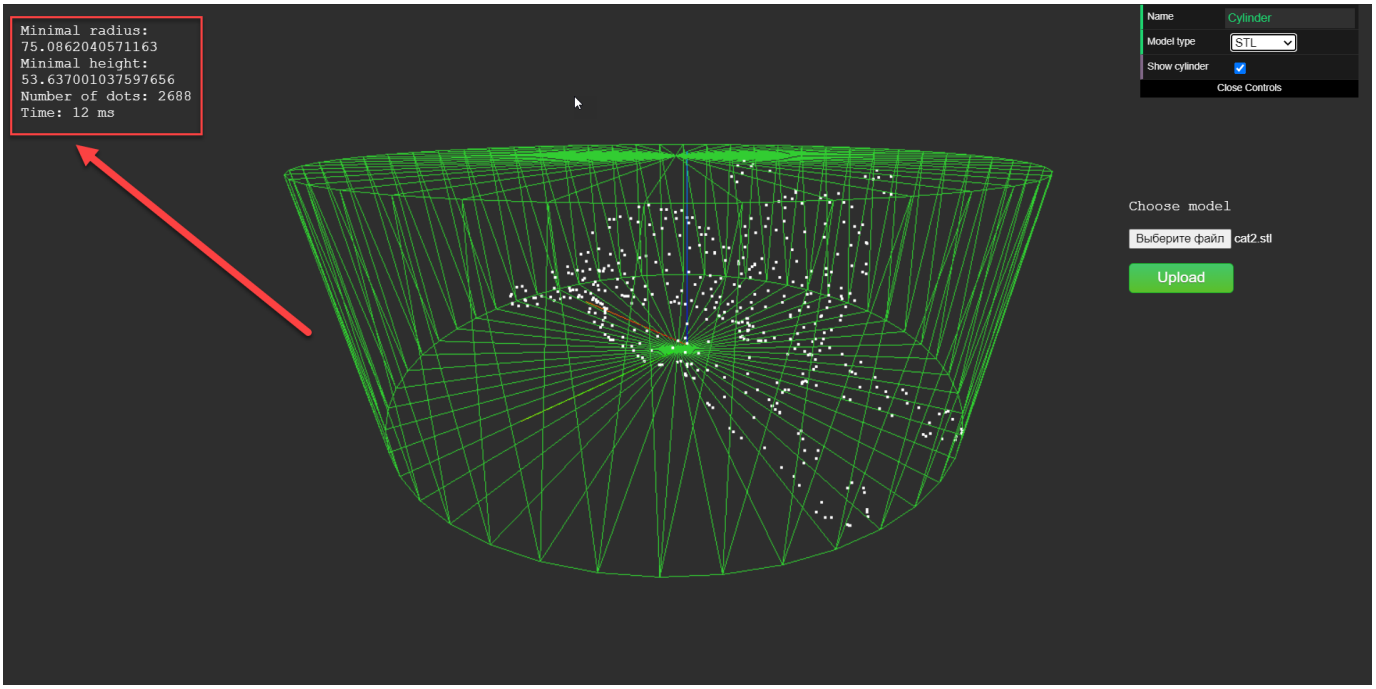

Рисунок 4.7 – Блок з результатами роботи алгоритму, кількістю точок та часом роботи алгоритму

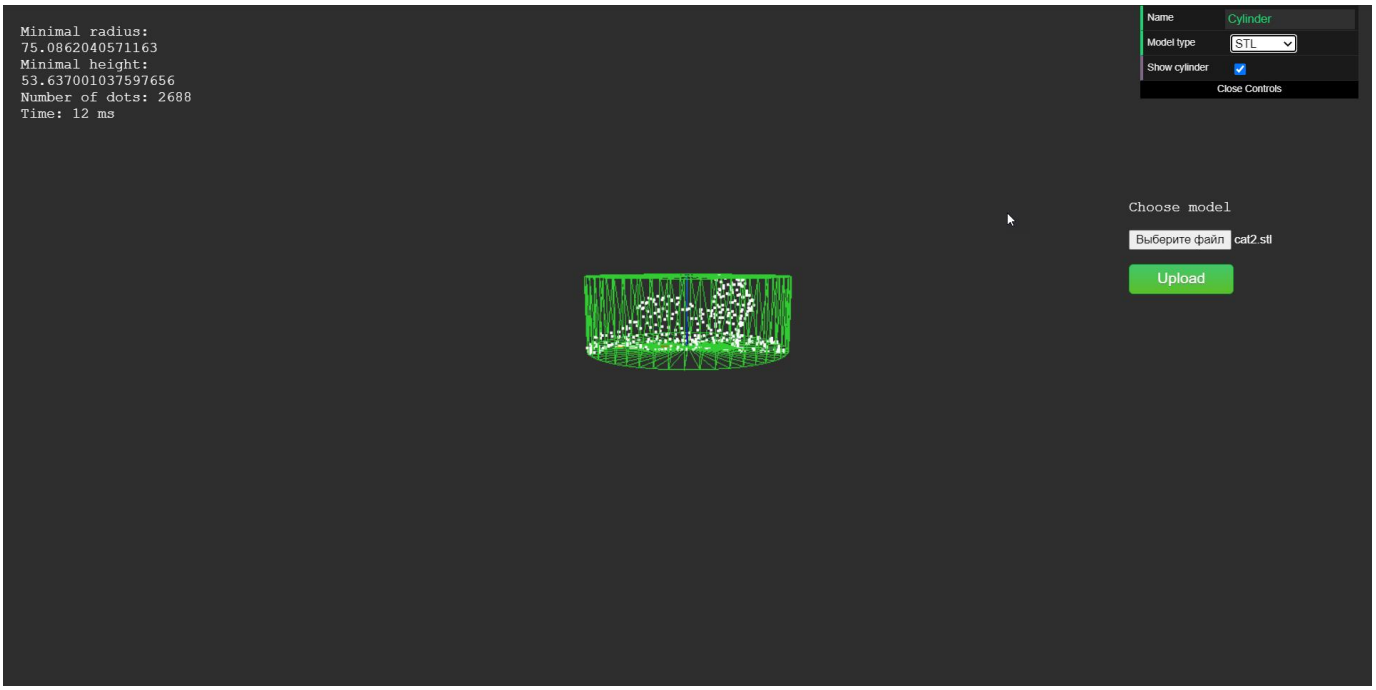

Рисунок 4.8 – Результат маніпулювання камерою для віддалення та обертання 3D моделі разом з обмежуючим циліндром

Недоліком розробленої системи є відсутність можливості зміни кута осі циліндру для розрахунку мінімального циліндру відносно інших проекцій на площину XOY.

Висновки до розділу 4

В результаті виконання роботи було створено веб-сервіс розрахунку мінімального габаритного циліндру 3D моделі на базі мікросервісної архітектури. Розроблено інтерфейс, що сприяє зручній взаємодії користувача з програмним продуктом і надає можливість керувати відображенням даних та переглядати отримані результати.

В даному розділі було розглянуто системні вимоги для оптимальної роботи з веб-додатком. Наведено сценарії роботи користувача з розробленим програмним забезпеченням. Було зазначено недолік розробленої системи.

ВИСНОВКИ

В бакалаврській роботі розв’язано наукове завдання перевірки належності 3D моделі заданому об'єму. При вирішенні поставлених задач були отримані наступні висновки:

1. Проведено огляд і аналіз систем, які включають в себе алгоритми розрахунку мінімальних обмежуючих тіл для хмари точок, в результаті чого обґрунтовано актуальність розробки системи для підрахунку мінімального габаритного циліндру.
2. Проаналізовано сучасні платформи для розробки веб-систем, та прийнято рішення про розгортання веб-системи на хмарній платформі Google Cloud з використанням мікросервісної архітектури, а в якості мови програмування обрано JavaScript (для клієнтської та серверної частини системи) з використанням оточення (рушія) Node.js.
3. Надано подальшого розвитку алгоритму Еммеріха Вельцля для пошуку мінімальної обмежуючої окружності зі складністю $O(n)$, що надало можливість проводити перевірку можливості виготовлення 3D моделі з циліндричної заготовки із заданими характеристиками.
4. Реалізовано веб-систему, що надало можливість завантажувати хмару точок з файлу 3D моделі у форматі STL. Проведено дослідження, які показали ефективність реалізованого алгоритму.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Project Nayuki [Електронний ресурс] — Режим доступу: https://www.nayuki.io/page/smallest-enclosing-circle.

2. Node.js About [Електронний ресурс] — Режим доступу: https://nodejs.org/en/about/.

3. MDN Web docs [Електронний ресурс] — Режим доступу: https://developer.mozilla.org/ru/docs/Web/JavaScript.

4. Хмарна піраміда: IAAS, PAAS I SAAS [Електронний ресурс] — Режим доступу: https://gigacloud.ua/blog/navchannja/hmarna-piramida-iaas-paas-i-saas.

5. Выбор стратегии деплоя микросервисов [Електронний ресурс] — Режим доступу: https://bool.dev/blog/detail/vybor-strategii-deploya-mikroservisov.

6. New Scheduling Strategies for Randomized Incremental Algorithms in the Context of Speculative Parallelization [Електронний ресурс] — Режим доступу: https://www.researchgate.net/publication/3049497_New_Scheduling_Strategies_for_Randomized_Incremental_Algorithms_in_the_Context_of_Speculative_Parallelization.

7. Хоган Б. HTML5 и CSS3. Веб-разработка по стандартам нового поколения / Б. Хоган – Пітер, видавництво Питер, 2014 – 318с.

8. Кантелон, М. Node.js в действии / М. Кантелон. – Пітер, видавництво Питер, 2015. – 810 с.

9. Документація Node.Js [Електронний ресурс] – Режим доступу: https://nodejs.org/uk/docs/.

10. HtmlBook [Електронний ресурс] – Режим доступу: http://htmlbook.ru/.

ДОДАТОК А

Мікросервіс розрахунку мінімального габаритного циліндру 3D моделі

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61_№61119_20Б

Аркушів 1

Київ 2020

Позначення	Найменування	Примітки
	Документація	
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61_№61119_20Б	Записка.docx	Текстова частина дипломної роботи
	Компоненти	
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61_№61119_20Б	server.js	Файл підключення клієнтської частини веб-сервісу до серверу
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61_№61119_20Б	algorithm.js	Файл з алгоритмом розрахунку мінімального габаритного циліндру
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61_№61119_20Б	dots.js	Файл генерації випадкових точок і парсингу STL моделі
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61_№61_№61119_20Б	index.html	Файл з клієнтською частиною системи

ДОДАТОК Б

Мікросервіс розрахунку мінімального габаритного циліндру 3D моделі

Лістинг програми

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61_№61119_20Б

Аркушів 10

Київ 2020

Текст форми відправки файлу і форми відправки кількості точок на сервер

```
<form action="http://storage.googleapis.com/model-bucket-cylinder"
  method="post" enctype="multipart/form-data" id="formSTL">
    <input type="hidden" name="key" value="${filename}" />
    <input type="hidden" name="success_action_status" value="204" />
    <p>
        <label for="file" id="label">Choose model</label>
    </p>
    <input type="file" name="file" id="file" accept="application/sla">
    <p>
        <input type="submit" value="Upload" id="stlb" class="myButton">
    </p>
</form>

<form id="formRandom" onsubmit="return false;">
    <p>
        <label id="label"> Number of dots to generate: </label>
    </p>
    <p>
        <input type="number" value="10" id="quantity" name="quantity"/>
    </p>
    <p>
        <input type="submit" id="calc" value="Upload" class="myButton"></button>
    </p>
</form>
```

Текст створення сцени та додавання контролеру

```
import { OrbitControls } from './js/OrbitControls.js';
import * as THREE from './js/three.module.js';
import { STLLoader } from './js/STLLoader.js';

var scene = new THREE.Scene();
scene.background = new THREE.Color( '#2E2E2E' );
var camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight,
0.1, 1000);
camera.up = new THREE.Vector3( 0, 0, 1 );

var renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);
```

```
var controls = new OrbitControls( camera, renderer.domElement );

controls.update();

camera.position.z = 5;
var animate = function () {
             requestAnimationFrame( animate );

    controls.update();
    renderer.render( scene, camera );
};

animate();
```

Текст запиту на сервер та візуалізації циліндру і точок

```
var status = function (response) {
    if (response.status !== 200) {
        return Promise.reject(new Error(response.statusText));
        alert("Something is wrong with your file!");
    }
    return Promise.resolve(response);
}

var text = function (response) {
    return response.text();
}

fetch('https://minimum-overall-cylinder.ew.r.appspot.com/welzl')
.then(status)
.then(text)
.then(function (data) {
    var result = JSON.parse(data);

    if (result.maxZ > 0 && result.minZ > 0 || result.maxZ < 0 && result.minZ < 0) {
        var height = Math.abs(result.maxZ) - Math.abs(result.minZ);
    }
    else {
        var height = Math.abs(result.maxZ) + Math.abs(result.minZ);
    }
```

```
    var axesHelper = new THREE.AxesHelper( 4 );
    axesHelper.name = "ax";
    scene.add( axesHelper );

    var info = document.getElementById('overlay').innerHTML = "Minimal radius:<br \/>" +
result.circle.r + "<br \/>Minimal height:<br \/>" + height + "<br \/>Number of dots: " +
result.numberOfDots + "<br \/>Time: " + result.time;

    var geometry = new THREE.CylinderGeometry( result.circle.r, result.circle.r, height,
50);
    var material = new THREE.MeshBasicMaterial( {color: 0x32CD32, wireframe: true} );
    var cylinder = new THREE.Mesh( geometry, material );

    cylinder.position.x = result.circle.x;
    cylinder.position.y = result.circle.y;
    cylinder.position.z = (result.maxZ + result.minZ) / 2;
    cylinder.rotation.x = Math.PI / 2;
    cylinder.name = "cyl";

    scene.add( cylinder );

    var group = new THREE.Group();
    group.name = "gdot";

    var finDots = JSON.parse(result.dots);
    for (var i = 0; i < finDots.dot.length; i++) {
        var counter = finDots.dot[i];
        var dotGeometry = new THREE.Geometry();
        dotGeometry.vertices.push(new THREE.Vector3( counter.x, counter.y, counter.z ));
        var dotMaterial = new THREE.PointsMaterial( { size: 3, sizeAttenuation: false }
);
        var dot = new THREE.Points( dotGeometry, dotMaterial );
        group.add( dot );
}
     scene.add( group );
});
```

Текст мікросервісу, що завантажує клієнтську частину програми

```
const port = process.env.PORT;
const express = require('express');

const app = express()
```

```
var connect = require('connect');
var serveStatic = require('serve-static');

connect().use(serveStatic(__dirname+'/res')).listen(port, function(){
    console.log('Server running');
});
```

Текст мікросервісу, що формує випадкові точки

```
const express = require("express");
const app = express();
const bodyParser = require("body-parser");
const fs = require("fs");

app.get('/random-dots', (req, res) => {
    let reqQuantity = req.query.quantity;
    var dots = {
        'dot': [],
        'state': true
    };

    function createDotsArray(rangeX1, rangeX2, rangeY1, rangeY2, rangeZ1, rangeZ2,
quantity) {
        var points = [];
        var rand1;
        var rand2;
        for(var i = 0; i < quantity; i++){
            randX = rangeX1 + Math.random() * (rangeX2 - rangeX1);
            randY = rangeY1 + Math.random() * (rangeY2 - rangeY1);
            randZ = rangeZ1 + Math.random() * (rangeZ2 - rangeZ1);
            points[i] = {
                x: randX,
                y: randY,
                z: randZ
            }
            dots.dot.push(points[i]);
        }
    }
    createDotsArray(-1, 1, 0, 1, -1, 1, reqQuantity)
    var myJsonString = JSON.stringify(dots);
```

```
    res.send(myJsonString);
});

app.listen(process.env.PORT, () => {
    console.log("Up and running (random dots service)");
});
```

Текст парсингу файлу формату STL

```
  var reqFile = req.query.file;

  const {Storage} = require('@google-cloud/storage');

  const bucketName='model-bucket-cylinder';
  const fileName=reqFile;
  const storage = new Storage();
  const file = storage.bucket(bucketName).file(fileName);

  var parseSTL = require('parse-stl');

  file.download({
      // don't set destination here
      }).then((data) => {
            //const contents = data[0];  // contents is the file as Buffer

            var mesh = parseSTL(data[0]);

            console.log(mesh);

            var dots = {
                  'dot': [],
                  'state': true
            };

            for (var i = 0; i < mesh.positions.length; i++) {
                  var x = mesh.positions[i][0];
                  var y = mesh.positions[i][1];
                  var z = mesh.positions[i][2];

                  points = {
                        x: x,
```

```
                        y: y,
                        z: z
                }
                dots.dot.push(points);
        }
        res.send(dots);
});
```

Текст алгоритму

```
function makeCircle(points) {
	// Clone list to preserve the caller's data, do Durstenfeld shuffle
	var shuffled = points.slice();
	for (var i = points.length - 1; i >= 0; i--) {
		var j = Math.floor(Math.random() * (i + 1));
		j = Math.max(Math.min(j, i), 0);
		var temp = shuffled[i];
		shuffled[i] = shuffled[j];
		shuffled[j] = temp;
	}

	// Progressively add points to circle or recompute circle
	var c = null;
	shuffled.forEach(function(p, i) {
		if (c === null || !isInCircle(c, p))
			c = makeCircleOnePoint(shuffled.slice(0, i + 1), p);
	});
	return c;
}

// One boundary point known
function makeCircleOnePoint(points, p) {
	var c = {x: p.x, y: p.y, r: 0};
	points.forEach(function(q, i) {
		if (!isInCircle(c, q)) {
			if (c.r == 0)
				c = makeDiameter(p, q);
			else
				c = makeCircleTwoPoints(points.slice(0, i + 1), p, q);
		}
	});
```

```
	return c;
}

// Two boundary points known
function makeCircleTwoPoints(points, p, q) {
	var circ = makeDiameter(p, q);
	var left  = null;
	var right = null;

	// For each point not in the two-point circle
	points.forEach(function(r) {
		if (isInCircle(circ, r))
			return;

		// Form a circumcircle and classify it on left or right side
		var cross = crossProduct(p.x, p.y, q.x, q.y, r.x, r.y);
		var c = makeCircumcircle(p, q, r);
		if (c === null)
			return;
		else if (cross > 0 && (left === null || crossProduct(p.x, p.y, q.x, q.y,
c.x, c.y) > crossProduct(p.x, p.y, q.x, q.y, left.x, left.y)))
			left = c;
		else if (cross < 0 && (right === null || crossProduct(p.x, p.y, q.x, q.y,
c.x, c.y) < crossProduct(p.x, p.y, q.x, q.y, right.x, right.y)))
			right = c;
	});

	// Select which circle to return
	if (left === null && right === null)
		return circ;
	else if (left === null && right !== null)
		return right;
	else if (left !== null && right === null)
		return left;
	else if (left !== null && right !== null)
		return left.r <= right.r ? left : right;
	else
		throw "Assertion error";
}
```

```
function makeDiameter(a, b) {
	var cx = (a.x + b.x) / 2;
	var cy = (a.y + b.y) / 2;
	var r0 = distance(cx, cy, a.x, a.y);
	var r1 = distance(cx, cy, b.x, b.y);
	return {x: cx, y: cy, r: Math.max(r0, r1)};
}

function makeCircumcircle(a, b, c) {
	// Mathematical algorithm from Wikipedia: Circumscribed circle
	var ox = (Math.min(a.x, b.x, c.x) + Math.max(a.x, b.x, c.x)) / 2;
	var oy = (Math.min(a.y, b.y, c.y) + Math.max(a.y, b.y, c.y)) / 2;
	var ax = a.x - ox,  ay = a.y - oy;
	var bx = b.x - ox,  by = b.y - oy;
	var cx = c.x - ox,  cy = c.y - oy;
	var d = (ax * (by - cy) + bx * (cy - ay) + cx * (ay - by)) * 2;
	if (d == 0)
		return null;
	var x = ox + ((ax*ax + ay*ay) * (by - cy) + (bx*bx + by*by) * (cy - ay) + (cx*cx
+ cy*cy) * (ay - by)) / d;
	var y = oy + ((ax*ax + ay*ay) * (cx - bx) + (bx*bx + by*by) * (ax - cx) + (cx*cx
+ cy*cy) * (bx - ax)) / d;
	var ra = distance(x, y, a.x, a.y);
	var rb = distance(x, y, b.x, b.y);
	var rc = distance(x, y, c.x, c.y);
	return {x: x, y: y, r: Math.max(ra, rb, rc)};
}

/* Simple mathematical functions */

var MULTIPLICATIVE_EPSILON = 1 + 1e-14;
//var MULTIPLICATIVE_EPSILON = 1;

function isInCircle(c, p) {
	return c !== null && distance(p.x, p.y, c.x, c.y) <= c.r *
MULTIPLICATIVE_EPSILON;
}

// Returns twice the signed area of the triangle defined by (x0, y0), (x1, y1), (x2,
y2).
// 2x area of triangle by 3 dots
function crossProduct(x0, y0, x1, y1, x2, y2) {
```

```
	return (x1 - x0) * (y2 - y0) - (y1 - y0) * (x2 - x0);
}

// Distance between two points
function distance(x0, y0, x1, y1) {
	return Math.hypot(x0 - x1, y0 - y1);
}

// (This is for old browsers)
if (!("hypot" in Math)) {  // Polyfill
	Math.hypot = function(x, y) {
		return Math.sqrt(x * x + y * y);
	};
}
```

Текст мікросервісу обрахунку мінімального габаритного циліндру і відправка результатів на сервер

```
const express = require("express");
const app = express();
const bodyParser = require("body-parser");
const request = require("request");

"use strict";

app.get('/welzl', (req, res) => {
	var dotNumb = 5 + Math.floor(Math.random() * Math.floor(20));
	const url = "https://minimum-overall-cylinder.ew.r.appspot.com/random-
	dots?quantity=" + dotNumb;
	request.get(url, (error, response, body) => {
		let json = JSON.parse(body);
		var result = {}
		result.dots = JSON.stringify(json);
		let zs = [];
		let zless = [];
		let temp;

		for (var i = 0; i < json.dot.length; i++) {
			var counter = json.dot[i];

			zs.push(counter.z);
```

```
                delete counter.z;

                zless.push(counter);
            }

            const start = new Date().getTime();
            result.circle = makeCircle(zless);
            const end = new Date().getTime();
            result.maxZ = arrayMax(zs);
            result.minZ = arrayMin(zs);
            result.numberOfDots = dotNumb;
            result.time = (end - start) + ' ms';

            res.send(result);
        });
}

app.listen(process.env.PORT, () => {
    console.log("Up and running (algorithm service)");
});
```

ДОДАТОК В

Мікросервіс розрахунку мінімального габаритного циліндру 3D моделі

Опис програмного коду

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61_№61119_20Б

Аркушів 8

Київ 2020

АНОТАЦІЯ

Даний додаток містить опис веб-сервісу розрахунку мінімального габаритного циліндру 3D моделі на базі хмарних технологій. Даний веб-сервіс реалізован на основі мікросервісної архітектури. Мікросервіси виконують наступні завдання:

- завантаження і формування хмари точок з файлу формату STL;
- формування хмари випадкових точок;
- розрахунок мінімального габаритного циліндру для хмари точок;
- візуалізація циліндру і хмари точок з можливістю взаємодії з ними (маніпулювання камерою);
- відображення результатів розрахунку, кількості точок та часу виконання алгоритму.

При розробці даного сервісу використовувалась мова розмітки документів HTML, таблиця каскадних стилів CSS, мова програмування JavaScript, середовище Node.js і провайдер хмарних обчислень Google Cloud.

ЗМІСТ

1. ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку міститься опис веб-сервісу пошуку мінімального габаритного циліндру 3D моделі. У додатку Б міститься програмний код головних частин розробленої системи.

Для роботи з розробленим додатком необхідно мати доступ до мережі Інтернет, веб-браузер з ввімкненим JavaScript та модель у форматі STL. При розробці програмного продукту використовувалась платформа Node.js, мова програмування JavaScript та середовище розробки Visual Studio Code.

Для реалізації мікросервісу розрахунку мінімального габаритного циліндру 3D моделі було розширено рандомізований алгоритм пошуку мінімальної обмежуючої окружності Еммеріха Вельцля зі складністю $O(n)$.

2. ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблені компоненти виконують завдання розрахунку мінімального габаритного циліндру завантаженої користувачем 3D моделі. Це відбувається за допомогою вершин трикутників, з яких складається модель.

Розроблений додаток може використовуватися на різноманітних підприємствах та виробництвах в якості програми, що дає змогу визначати можливість виготовлення моделі з циліндричної заготівки.

Функціональні обмеження на використання додатку полягають у форматі завантаженої моделі.

3. ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Під час запуску сервісу перед користувачем з’являється головне вікно з візуалізацією циліндру для випадкової кількості випадкових точок.

Чекбокс “Show cylinder” відповідає за відображення циліндру.

Після вибору джерелом завантаження точок файлу на екрані відображається форма з вибором файлу з локальної директорії системи. Файл не може перевищувати 9.216 MB, в інакшому випадку виводиться попередження про неможливість обробки файлу такого розміру.

У випадку вибору джерелом завантаження точок генерацію випадкових точок на екрані відображається форма з введенням необхідної кількості точок. Кількість точок не може перевищувати 1000000.

Кнопка “Upload” відповідає за відправку даних на сервер та запуск алгоритму.

У верхній лівій частині екрану знаходиться блок з результатами роботи алгоритму, кількістю точок та часом роботи алгоритму.

4. ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для забезпечення повноцінної роботи та досягнення високої точності роботи хмарного сервісу розрахунку мінімального габаритного циліндру було обрано провайдер хмарних обчислень Google Cloud. Для написання серверної частини веб-сервісу було обрано середовище Node.js, використання якого дозволяє максимально просто втілити архітектурний стиль REST, а також має досить швидку обробку вхідних запитів користувача.

Також для роботи з 3D графікою у браузері використовувалися додаткові JavaScript бібліотеки Three.js та OrbitControls.js.

Розроблений додаток працює у браузері з увімкненим JavaScript та доступом до мережі Інтернет. Для роботи з ним користувачу необхідно мати 3D модель у форматі STL, розрахунок мінімального габаритного циліндру якої необхідно зробити.

5. ВИКЛИК І ЗАВАНТАЖЕННЯ

Розроблена система не потребує додаткової інсталяції. Для того, щоб з нею працювати потрібно лише доступ до мережі Інтернет, веб-браузер з увімкненим JavaScript та модель для розрахунку витрат матеріалу у форматі STL.

Після запуску користувач переходить до головної сторінки сервісу де він має змогу обрати необхідний файл з моделлю.

6. ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними є файл 3D моделі формату STL або кількість випадкових точок для генерації.

Вихідними даними є результат роботи алгоритму (мінімальний радіус і висота) та кількість точок і час роботи алгоритму.

ДОДАТОК Г

Мікросервіс розрахунку мінімального габаритного циліндру 3D моделі

Апробації

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61_№61119_20Б

Аркушів 3

Київ 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

СУЧАСНІ ПРОБЛЕМИ НАУКОВОГО ЗАБЕЗПЕЧЕННЯ ЕНЕРГЕТИКИ

Матеріали XVIII Міжнародної науково-практичної конференції молодих вчених і студентів 2020 року

ТОМ 2

Київ- 2020

УДК 004.94:514.18

Студент 4 курсу, гр. ТР-61 Світла Л.В.
Доц., к.т.н. Демчишин А.А.

МІКРОСЕРВІС РОЗРАХУНКУ МІНІМАЛЬНОГО ГАБАРИТНОГО ЦИЛІНДРУ 3D МОДЕЛІ

На сьогоднішній день в умовах високого темпу виробництва на підприємствах можуть виникати труднощі при виготовленні певних нестандартних деталей. Метою даної системи є вирішення проблеми виготовлення деталей із циліндричних заготівок найбільш оптимальним способом.

Оскільки пошук мінімального циліндру не є стандартною задачею, то її необхідно розбити на декілька менших задач. Зробивши це можна прийти до висновку, що пошук мінімального циліндру відноситься до класу LP-повних задач. Це означає, що дану задачу можна вирішити за лінійний час, якщо підзадача в рамках основної задачі вирішується за лінійний час. Пошук мінімального циліндру можна поділити на 2 підзадачі: пошук мінімальної висоти циліндру і пошук мінімальної окружності. Для того, щоб звести задачу до лінійного рішення, пошук мінімальної висоти можна вирішити заданням певного параметру (направлення). Тоді основну проблему становить пошук мінімальної окружності. Задача пошуку мінімальної окружності за лінійний час була вирішена за допомогою рандомізованого рекурсивного алгоритму Еммеріха Вельцля, який був оснований на алгоритмі лінійного програмування. Цей алгоритм полягає в обчисленні найменшої окружності, що обмежує об'єднання множини точок S і Q, якщо будь-яка точка множини Q є граничною точкою можливої обмежувальної окружності [1].

Програмна система має надавати можливість обрахунку мінімального габаритного циліндру після загрузки відповідного файлу з хмарою точок. Для реалізації веб-додатку було обрано платформу node.js разом із фреймворком express, а для відображення результатів обрахунку – бібліотеку tree.js. Для зручності розробки та подальшого розширення системи було обрано мікросервісну архітектуру. Основні модулі програмної системи наступні:

- мікросервіс генерації хмари точок;
- мікросервіс обрахунку мінімального габаритного циліндру;
- модуль графічного інтерфейсу.

Розроблений програмний додаток має інтуїтивно зрозумілий і гармонійно побудований інтерфейс, що надає можливості використання системи спеціалістами з різними рівнями підготовки. Окрім того, програма має можливість генерувати випадкові дані з можливістю їх корегування для тестування системи.

Дана програмна система вирішує проблему оптимізації витрат матеріалу для виготовлення нестандартних деталей із циліндричних заготівок. Окрім того, завдяки додатку, отримано можливість автоматичного розрахунку спроможності виготовлення деталі за заданими параметрами. Система надає підгрунтя до збереження матеріальних ресурсів та часу робітників і клієнтів підприємства.

Перелік посилань:

1. Сайт ПОИВС // Задача о наименьшей окружности [Електронний ресурс] – Режим доступу: http://poivs.tsput.ru/ru/Math/Geometry/ComputationalGeometry/LeastCircleProblem